CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Design Flaws as Security Threats

Master's thesis in Computer Science and Engineering

Danial Hosseini
Kyriakos Malamas

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY AND UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2017

Master's thesis 2017:06

Design Flaws as Security Threats

A Master Thesis in Software Engineering

DANIAL HOSSEINI
KYRIAKOS MALAMAS

UNIVERSITY OF
GOTHENBURG

Department of Computer Science and Engineering
Software Engineering and Technology
Chalmers University of Technology and University of Gothenburg
Gothenburg, Sweden 2017

Design Flaws as Security Threats
A Master Thesis in Software Engineering
Danial Hosseini
Kyriakos Malamas

© Danial Hosseini, Kyriakos Malamas 2017.

Supervisor: Riccardo Scandariato, Computer Science and Engineering.
Examiner: Jan-Philipp Steghöfer, Computer Science and Engineering

Master's Thesis 2017
Department of Computer Science and Engineering
Software Engineering and Technology
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by [Chalmers University of Technology and University of Gothenburg]
Gothenburg, Sweden 2017

Design Flaws as Security Threats
A Master Thesis in Software Engineering
Danial Hosseini - danialh@student.chalmers.se
Kyriakos Malamas - malalas@student.chalmers.se
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Current practices used to evaluate security in the architecture, usually involve threat modeling activities. These activities, however, often require a significant amount of resources, in terms of time and effort, to achieve complete threat coverage. Other approaches focus on identifying design flaws early on and usually involve the definition of a rule set and the creation of detailed models to check against these rules which require a significant amount of effort and expertise. This study proposes a manual approach in the form of a catalog of design flaws along with detection guidelines in order to detect design flaws related to security threats. A descriptive study is conducted to evaluate the effectiveness and productivity of the approach, as well as how the detected flaws can be related to threats identified by STRIDE. Finally, further investigation is done to understand how the approach can complement existing threat modeling techniques.

Keywords: Catalog of design flaws, Design flaws, Security, Secure architecture, Threat modeling.

Acknowledgements

We would like to thank our supervisors Riccardo Scandariato and Katja Tuma from Chalmers University of Technology, for their valuable time and input along with their expertise that assisted us in completing this research. In addition, we are also immensely grateful to Atul Yadav and Christian Sandberg at Volvo Group Trucks Technology for offering their insights and the necessary resources to complete this research. Finally, we would also like to thank the students from Chalmers University, as well as the practitioners from Volvo Group Trucks Technology for participating in the study and taking the time to answer our questionnaire.

Danial Hosseini and Kyriakos Malamas, Gothenburg, June 2017

Contents

List of Figures x

List of Tables xi

1 Introduction 1
- 1.1 Research questions . . . 2
- 1.2 Scientific contribution . . . 2
- 1.3 Methodology . . . 3
- 1.4 Thesis outline . . . 3

2 Considering Security in Design 5
- 2.1 Software design and design flaws . . . 5
  - 2.1.1 Design Principles . . . 5
  - 2.1.2 Design and Security Patterns . . . 6
  - 2.1.3 Design Flaws . . . 7
- 2.2 Current research approaches on identification of design flaws . . . 8
- 2.3 Threat Modeling and Risk Assessment Methods . . . 10
  - 2.3.1 STRIDE and DREAD . . . 10
  - 2.3.2 TRIKE . . . 11
  - 2.3.3 Use Cases and Misuse Cases . . . 11
  - 2.3.4 OCTAVE . . . 12
  - 2.3.5 PASTA . . . 13
  - 2.3.6 TVRA . . . 13
  - 2.3.7 EVITA - THROP . . . 14
  - 2.3.8 HEAVENS Security Model . . . 15

3 Compiling the Catalog 16
- 3.1 Defining the scope of design flaws . . . 17
- 3.2 Data collection . . . 17
  - 3.2.1 Threat and Vulnerability Taxonomies . . . 17
  - 3.2.2 Vulnerability Databases . . . 19
  - 3.2.3 Other Resources . . . 19
  - 3.2.4 Related Work Regarding Catalogs of Design Flaws . . . 20
- 3.3 Deriving a list of most common threats and weaknesses. . . . 21
- 3.4 Compile the catalog of design flaws. . . . 22
- 3.5 Defining detection guidelines. . . . 23

4 Evaluation of the Catalog **25**
4.1 Preparation for the evaluation . . . 25
4.1.1 Selected measurements and metrics . . . 25
4.1.2 Pilot run . . . 26
4.2 Evaluation with students . . . 27
4.2.1 Experimental object . . . 27
4.2.2 Participants . . . 27
4.2.3 Task . . . 27
4.2.4 Establishing a Baseline . . . 28
4.3 Relating design flaws to threats . . . 29
4.4 Evaluation at Volvo Group Truck Technologies . . . 29

5 Results **31**
5.1 Results from students . . . 31
5.2 Relating design flaws to threats derived from STRIDE . . . 33
5.3 Results from Volvo . . . 36

6 Discussion **38**
6.1 Effectiveness of the catalog . . . 38
6.2 Productivity of the analyst using the catalog . . . 38
6.3 How the catalog can complement threat modeling techniques . . . 39
6.4 Using the catalog in the automotive industry context . . . 39
6.5 Similar Studies . . . 40
6.6 Threats to Validity . . . 40
6.6.1 Construct Validity . . . 40
6.6.2 Internal Validity . . . 41
6.6.3 External Validity . . . 42
6.7 Future Work . . . 42

7 Conclusion **43**

References **45**

A Catalog of Design Flaws **I**

B Results from students **XII**

C STRIDE threat comparison with design flaws baseline **XIV**

D Questionnaire **XXIV**

E Documents provided in the lab package **XXVIII**
E.1 HomeSys Domain Problem Description . . . XXVIII
E.2 HomeSys Requirement Specification . . . XXXIX
E.3 HomeSys Architectural Description . . . LII

List of Figures

1.1 The software development lifecycle and its different phases. 1
1.2 Steps of the methodology throughout the study. 3

2.1 OCTAVE Phases 13
2.2 Workflow of HEAVENS security model 15
2.3 HEAVENS Security Level (SL) 15

3.1 Overview of the process to compile the catalog. 17

5.1 Number of flaws detected in HomeSys per catalog entry. 32

List of Tables

2.1 Design Principles presented by Saltzer and Schroeder 5
2.2 Design Principles presented by Viega and McGraw 6
2.3 Element types mapped to STRIDE threat categories. 10

3.1 Part of threats and weaknesses collected in step 3 and related CWE entries. 21
3.2 Design flaws to be included in the catalog. 23

4.1 Baseline of flaws for HomeSys. 29

5.1 Student results and measurements 31
5.2 Threats from STRIDE analysis and related flaws in the baseline . . . 34
5.3 Number of design flaws in baseline and student results associated with each threat 35
5.4 Number of threats for which associated flaws were found. 36
5.5 Results from Volvo. 37

B.1 Baseline of flaws for HomeSys. XIII

1
Introduction

The importance of integrating security has emerged as a leading factor for many worldwide organizations. As software systems increase in size, complexity, and new technologies surface, security is becoming all more important [1]. The potential cost connected with security incidents and the continued operational costs associated with keeping up to date with security patches requires organizations to take action and give more consideration on how to address potential security threats [1]. Currently, most organizations focus on addressing software security in later stages of the software development lifecycle (SDLC). This is usually performed in the implementation phase Figure 1.1, where static analysis tools are used to identify vulnerabilities. Although this approach can help diminish vulnerabilities, performing security activities later can lead to increased costs, since the longer a vulnerability lies dormant, the more expensive it can be to fix [2], [3].

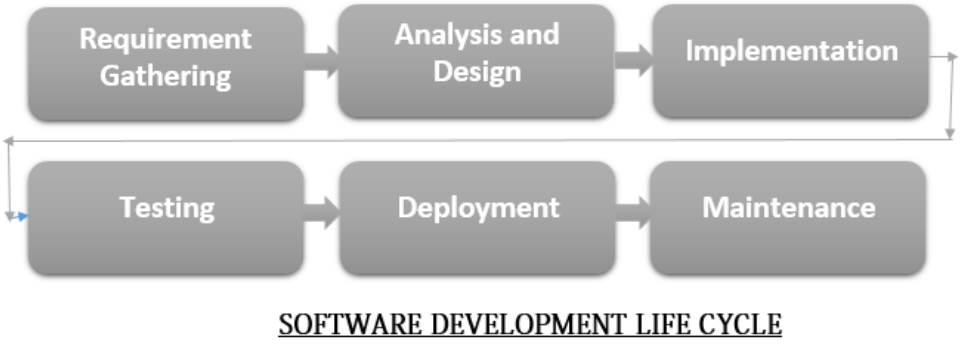


Figure 1.1: The software development lifecycle and its different phases.

A vulnerability in a system can be introduced through different sources, as it can be caused by design weaknesses, implementation errors, and system configuration errors, however, a large part of the vulnerabilities are caused by decisions made earlier in the SDLC [1]. Empirical data claim that 50% of the vulnerabilities are caused by architectural design flaws, which means that security needs to be considered earlier in the SDLC [1]. A design flaw can be an incorrect design decision that does not comply with the best practices or design principles normally used to tackle with recurring problems [4], [5]. A design flaw represents an error in the architecture and tends to be harder to eliminate if discovered too late in the SDLC, they also take more time and resources to resolve [1].

Commonly used practices for evaluating security in architecture, usually involve

threat modeling activities during the architecture and design phases. However, these activities are usually difficult to perform during early design stages, when specific implementation technologies have not been chosen yet [6]. Other approaches, focus on identifying design flaws early on and usually involve the definition of a rule set and the creation of detailed models to check against these rules, such the ones proposed by Berger et al. [7] and Almorsy et al. [8]. Although these approaches aim towards automating the process, they rely on the creation of a knowledge base of rules and elaborate models, which require a deep understanding of the domain and security as well as a significant amount of time and effort to create these models and capture all the required information.

Thus, our approach is aimed towards creating a catalog of design flaws along with detection guidelines to identify architectural design flaws in early design phases that can lead to security threats, therefore the primary focus will be on design flaws related to high abstraction levels of the architecture. The aim of the approach is not to replace any of the existing techniques, but rather to be used as a complementary method for practitioners to detect flaws related to security threats and mitigate those threats by addressing these flaws. The catalog of design flaws will be evaluated in a practical environment by students from Chalmers University of Technology and practitioners at Volvo Groups Trucks Technology.

This study was done in collaboration with Volvo Groups Trucks Technology under the scope of HoliSec (Holistic Approach to Improve Data Security) project. The goal of HoliSec is to develop integration-ready security development processes that are aligned with existing safety processes, verification and validation methods that address security, investigate security mechanism properties, and look at methods to secure both wired and wireless communication. This can be achieved by making sure security requirements are visible and influence all steps in the development chain, all the way from early development to late testing phases [9].

1.1 Research questions

For the purpose of this study the following research questions have been defined:

- RQ1: What is the effectiveness of the approach?
- RQ2: What is the productivity of the analysts using the approach?
- RQ3: How can this approach complement current knowledge-based threat modeling techniques?

1.2 Scientific contribution

This study proposes a catalog of design flaws that are related to security threats, to be used in the evaluation of system's architecture in early stages. Each flaw listed in the catalog can be detected by following the provided guidelines. Unlike other

approaches, it doesn't require elaborate models or expertise in the security domain to be able to use it. The approach is evaluated in a practical environment, by students and practitioners, in terms of effectiveness and productivity. Additionally, the study investigates how such an approach can complement existing threat modeling activities.

1.3 Methodology

The methodology for this thesis is described in Figure 1.2. The first step in the process starts by identifying and understanding the problem to establish the objective of the study. The second step in our methodology was to gather relevant data from different sources that could be used to form the catalog. The sources include threat and vulnerability taxonomies, online resources from related organizations such as Open Web Application Security Project (OWASP) Top 10 projects [1] and SANS 25 [2], databases such as Common Weakness Enumeration (CWE) [3] and any related work. A more detailed description on the resources used will be provided in section 3.2. The next step was to use the gathered data and compile the catalog of existing design flaw along with description and detection guidelines that assist in the identification of architectural design flaws. The fourth step involves the evaluation of the proposed catalog of design flaws with students from Chalmers and practitioners at Volvo. At the final step of the process, all results were documented and further work was discussed.

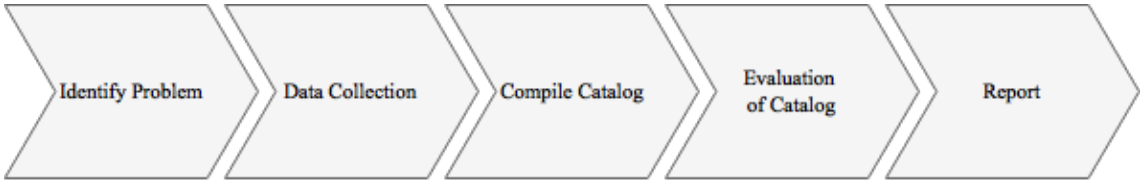


Figure 1.2: Steps of the methodology throughout the study.

1.4 Thesis outline

The remaining of this report is structured as follows. Chapter 2 summarizes existing definitions and terminologies for design principles, security patterns, and design flaws along with an overview of the existing approaches on identifying design flaws and threat modeling techniques. In Chapter 3 an overview of the process to compile the catalog is presented along with the data sources used to gather relevant data. Chapter 4 describes the planning of the evaluation along with the task given to students from Chalmers University of Technology and practitioners from Volvo Group

[1] OWASP Top 10 presents critical security issues existing in web applications in a yearly basis. `https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project`

[2] SANS 25 presents the most dangerous software errors. `https://www.sans.org/top25-software-errors/`

[3] CWE maintains a collection of common software security weaknesses. `https://cwe.mitre.org/data/index.html`

Trucks Technology (GTT). Chapter 5 describes how the results were processed as well as observations made during the evaluation. Chapter 6 discusses the results and how these can be used to answer the research questions in the study, along with threats to validity and suggestions for future work. Finally, Chapter 7 summarizes the findings of the study.

2
Considering Security in Design

2.1 Software design and design flaws

This section presents definitions and terminologies for design principles, security patterns, and design flaws according to literature.

2.1.1 Design Principles

Design principles are specific guidelines that can be followed in order to design a secure system. The principles demonstrate what to consider in order to enhance security as well as to aid in the development of a new system. This section provides an overview of the design principles proposed in the literature.
In 1975, Saltzer and Schroeder [10] presented a series of design principles for secure systems, which apply especially to protection mechanisms. The following design principles were proposed and can be seen in Table 2.1

Design Principle	Description
Economy of mechanism	A simpler design is easier to test and therefore the design should be as simple and small as possible.
Fail-safe defaults	Access should be denied by default and only be permitted when explicit permission exists.
Complete mediation	Every access to every object must be checked for authority.
Open design	The security of a mechanism should not depend on the secrecy of its design or implementation.
Separation of privilege	Access should be granted based on more than one piece of information. Protection mechanism that requires two keys are more robust and flexible.
Least privilege	Every process and user should be given the least set of privileges that it needs in order to complete its task.
Least common mechanism	The protection mechanism should be shared as little as possible among users.
Psychological acceptability	The protection mechanism should be easy to use.

Table 2.1: Design Principles presented by Saltzer and Schroeder

Aiming towards improving secure software development Viega and McGraw [1] present their own set of design principles, which has similarities to those proposed

by Saltzer and Schroeder [10]. Table 2.2 refers to the design principles introduced by Viega and McGraw [1]

Design Principle	Description
Secure the weakest link	The level of security is only as strong as the weakest link in the system.
Practice defense in depth	Use multiple complementary security mechanisms, so that a failure in one does not mean total insecurity.
Fail securely	Design the system so it fails in a secure manner.
Follow the principle of least privilege	Only the minimum access necessary to perform an operation should be granted, and that access should be granted only for the minimum amount of time.
Compartmentalize	Segment a system into multiple components that are protected independently to reduce the damage of an attack.
Keep it simple	Avoid unnecessary complexity by keeping the system as simple as possible.
Promote privacy	Promote privacy for the users and the system.
Remember that hiding secrets is hard	This principle assumes that even the most secure systems are amenable to inside attacks.
Be reluctant to trust	Instead of making assumptions that need to hold true, you should be reluctant to extend trust.
Use your community resources	Use well-known community resources that have been widely scrutinized and used.

Table 2.2: Design Principles presented by Viega and McGraw

Kenneth and Van Wyk [11] proposed an extensive list of 30 security architecture principles that are based on previous work by [10] and [1]. The collection ranges from specific principles, for example “fail safely” and "build in multiple layers of defense" to more generic design guidelines such as “remember to ask, what did I forget?”.

The collection of design principles often overlaps each other, and may not always be helpful when it comes to concrete design problems. However, they can further the understanding of which aspects are relevant to secure software development. Additionally, they convey the message that security should not be ignored since it’s an important factor of today’s software development [12].

2.1.2 Design and Security Patterns

Design patterns propose a generic solution to recurring design problems within a given context in software engineering. The concept of patterns was first used by Christopher Alexander in buildings architecture and similarities could be seen within software architecture resulting in further study and publications of patterns. Design Patterns are widely used in today’s software development and further research has been done in many domains, including the security domain [12].

Schumacher and Roedig provide the following definition: *"A Security Pattern describes a particular recurring security problem that arises in specific contexts and presents a well-proven generic scheme for its solution. A Security Pattern System is a collection of security patterns, together with guidelines for their implementation, combination and practical use in security engineering."* [13]. Their understanding is similar to the definition found at [14].

One of the first papers that addressed security patterns was published by Yoder and Baralow [15], with the focus on building secure software by proposing seven security patterns. Anand et al. provide a categorisation of security patterns and correlate them with vulnerability types [16]. Alvi and Zulkernine [17] proposed a classification scheme for security patterns based on the natural cause of security breaching in software applications. Hafiz et al. [18] provide different categorizations for security patterns that they applied while developing a pattern language. Alvi and Zulkernine [19] provide a comparison of the several classifications of security patterns and finally, Kienzle et al. [20], present a repository of security patterns.

2.1.3 Design Flaws

Apart from the term design flaw, there are a number of terms defined in the literature. It can also be referred to as architectural bad smells [21], design pattern defect [22] or architectural flaw [23].

Garcia et al. [21] define architectural bad smell as "*a commonly used set of architectural design decisions that negatively impacts system lifecycle properties, such as understandability, testability, extensibility, and reusability*". They also mention that architectural smells are analogous to code smells, since they both represent common solutions which are not necessarily wrong but can still have a negative effect on the overall quality of the software [21].

Moha et al. [22] define design pattern defects as occurring errors in the design of software that come from the absence or bad use of patterns. They distinguish between the defects as distorted, deformed, missing or excess design patterns.

According to Mirakhorli [23], an architectural flaw is a result of using inappropriate design choices in earlier stages of software development, incorrect implementation of security patterns or degradation of security architecture over time.

Andrade et al. [24] describe that architectural smells are structural attributes that can affect lifecycle properties, such as testability, understandability, and reusability. They can also affect quality properties, such as reliability and performance. According to Garcia et al. [21], this can occur by *"applying a design solution in an inappropriate context, mixing combinations of design abstractions that have undesirable emergent behaviors, or applying design abstractions at the wrong level of granularity"*.

By examining the definitions mentioned, we can conclude that a design flaw can affect the quality properties of the system and can be caused by an incorrect implementation (or omission) of a design pattern or failure to apply design principles properly. It can also be due to design solutions applied, that although tested before, they are not applicable in a specific context. Finally, as a system evolves, its architecture tends to degrade if its not designed in a way to consider future changes.

2.2 Current research approaches on identification of design flaws

According to Almorsy et al. [8] existing efforts to assess and evaluate the security of software architecture can be classified into two main categories, scenario based architectural analysis and metrics-based approaches. While scenario based analysis is focusing on generating a set of evaluation scenarios which is based on security requirements, metrics-based approaches focus on developing metrics that can be used to assess the software architecture. Examples of scenario based approaches, such as misuse cases or Data Flow Diagrams (DFDs) for use case scenarios, are [7], [25] and [26], while for metrics-based [27].

Almorsy et al. [8] introduce an automated risk analysis approach that uses architectural security scenarios and metrics. They use OCL to formalize attack signatures and to specify security metrics, in order to assess the security and architecture of the system. These OCL rules are then applied to the two UML models they create, *System Description Model (SDM)* and *Security Specification Model (SSM)* with the help of a tool. The knowledge-base of rules is based on CAPEC [28] and is limited on detecting 4 specific types of attacks in the system under analysis, listed below [8]:

- Man In The Middle (MITM) Attacks: The attacker intercepts communication between two components and makes independent connections with them relaying the messages, while the components think they communicate directly. The signature of this attack is insecure communication channels or communication in an untrusted zone.
- Denial of Service (DoS) Attacks: The goal of this attack is to make a system or a critical resource of the system unavailable to its legitimate users. This is usually done by overwhelming the system with requests, targeting publicly accessible components that lack proper authentication or data validation mechanisms.
- Data Tampering Attacks: This attack refers to tampering with the data at rest, transit or even during processing. This can be the result of the data being stored or transmitted in insecurely (in plaintext format), processing occurring in an untrusted environment or if authorization policy and access control allows unauthorized people to access data.
- Injection Attacks: If a system doesn't have a proper input validation mechanism, it is vulnerable to these types of attacks. The goal is to pass malicious input to a system to gain higher privileges, tamper with data or even cause a

system crash.

As mentioned in [8], developing security scenarios to be used in architecture evaluation is a very complicated task and requires deep knowledge of the security domain. Although their approach achieves a precision rate of 90% and a recall rate of 89%, according to them, the results of the analysis tool depends on the soundness of the metrics and scenario specified in OCL.

Another similar approach is proposed by Berger et al. [7] where they automatically extract threats from *Extended Data Flow diagrams (EDFDs)*. In order to achieve this, they use EDFDs, which cover all concepts of traditional DFDs but allow the use of additional semantics, and they create a knowledge base of rules and predefined patterns that will be checked against the EDFD using a rule checker in order to produce a threat model. Their approach supports 25 security related rules which mostly consist of entries from CWE [29] and CAPEC [28]. Their focus is not only in architectural issues, as according to [7] they also include rules concerning implementation issues, such as buffer overflow. More importantly, the creation of the knowledge-base requires security expertise and the EDFD needs to be done by a system expert.

Both of the approaches mentioned above aim towards an automated analysis of the architecture. In order to achieve these, both require detailed models to be constructed and a knowledge base of rules to be defined. However, defining security scenarios and metrics is a rather complex task and often requires a deep knowledge of the security domain. Additionally, the effectiveness of such tools is dependent on the knowledge base of rules and how often it is updated. Even then, only currently known attacks and vulnerabilities are covered.

Other approaches, such as the ones presented by D'Ambros et al. in [27] and Bunke and Sohr in [30], involve reverse engineering of the source code using detection strategies derived from design rules, guidelines and heuristics, in order detect design flaws. However, these design flaws mostly refer to OOP flaws and are not necessarily related to security. In [30], UML diagrams are derived from the source code in order to detect security patterns used. Although, this approach could be useful for maintenance, it is not in the scope of our study, since our focus is on how to detect any design flaws in the design phase and not after the development of the system.

The International Organization for Standardization is currently working on ISO/IEC PDTS 19249 [31] with the intention of publishing a catalog of architectural and design principles for secure products, systems, and application [31]. During the time our research was conducted, the standardization was still under development, and therefore conclusions could not be drawn.

2.3 Threat Modeling and Risk Assessment Methods

Threat modeling is considered as one of the important activities a security analyst can perform during the software development lifecycle to identify, quantify and address security threats associated with the system [1]. In this section, a number of threat modeling and risk assessments methods are presented.

2.3.1 STRIDE and DREAD

STRIDE is a threat modeling technique developed by Microsoft that supports evaluation of security throughout several activities [32]. The threat modeling process starts by modeling a Data Flow Diagram (DFD) to present how the data travels in the system. The DFD consists of elements, which are distinguished in data flows (DF), data stores (DS), process nodes (PN) and external entities (EE) [32]. The DFD is then reviewed, and threats based on the types of elements is organized according to the STRIDE threat categorization:

- *Spoofing* refers to an attacker impersonating another user or process.
- *Tampering* refers to when an attacker illegally modifies data in transit or rest.
- *Repudiation* refers to the absence of logged activities.
- *Information disclosure* refers to exposure of confidential information to those without access.
- *Denial of service* refers to a threat making the system temporary or permanently unavailable.
- *Elevation of privilege* refers to when an attacker obtains higher privileges that they are not entitled to.

The element types in the DFD can be mapped to more than one STRIDE categorization, as shown in Table 2.3.

	S	T	R	I	D	E
EE	x		x			
DF		x		x	x	
DS		x		x	x	
PN	x	x	x	x	x	x

Table 2.3: Element types mapped to STRIDE threat categories.

For each mapping between a generic threat and a DFD element type, STRIDE provides a hierarchical threat tree that consists of concrete threats that need to be considered. Currently, 15 threat trees are provided where each leaf represents a threat that can realize the root node [32].

STRIDE was often used in conjunction with DREAD, a risk assessment model created by Microsoft to quantify, compare and prioritize the risk presented by each evaluated threat. The DREAD acronym is formed from the first letter of each category:

- *Damage Potential* - How much damage will be done if the threat is exploited?
- *Reproducibility* - How easy is it to reproduce the threat exploit?
- *Exploitability* - How hard is it to exploit the threat?
- *Affected users* - What users will be affected if the threat is exploited?
- *Discoverability* - How easy is it to discover this threat?

A given threat is assessed using DREAD, each category is assigned a number between 0-10, higher numbers, indicate greater risk. Finally, the risk value for each threat is computed by calculating the average of all five categories. The decision to retire the DREAD model was made in 2008 by Microsoft since they discovered that the ratings lacked consistency and that DREAD was not an enough detailed risk methodology.

2.3.2 TRIKE

TRIKE is an open source threat modeling framework that started during 2006 and is still under development [33]. TRIKE is used to satisfy the security auditing process from a risk management perspective. The initial step of TRIKE is to compose a *requirements model* that is designed to provide an overview of what the system is intended to do and what assets are associated with the actions taken by the actors. The *requirement model* is used to ensure the level of risk assigned to each asset is classified as acceptable.

The next step of the methodology is to create a *implementation model* by modeling a DFD and specifying more implementation details for each element type, such as what libraries, platform, protocols and what type of data stores are used [33]. The *implementation model* is then examined to identify potential threats that are categorized into two different categories, either denial of service or elevation of privilege. The denial of service threat is when a privileged user is prevented from executing its intended actions. Elevation of privilege refers to when a user performs an activity beyond the initial permissions specific to that user. These threats are then used to build an attack graph to see what steps an attacker could take to realize a threat. Finally, countermeasures are suggested, and a risk model is generated based on assets, roles, actions and threat exposure [33].

2.3.3 Use Cases and Misuse Cases

Use cases are derived from requirements and specify the normal behavior and usage of the system usually in the form of structured scenarios, diagrams or stories. In addition, use cases are used at the concept phase to define the interaction between the actors and the system [34]. Misuse cases also referred as abuse cases or negatives scenarios is a well-known threat modeling technique with the primary objective of identifying undesirable behaviors between the system and actors via brainstorming of security experts to elicit security requirements. The brainstorming session involves the creation of misuse case diagrams together with the corresponding use case diagram containing the sequence of actions that can be performed to harm the system and the actors that could potentially initiate the attack [35]. Sindre and Opdahl [34] propose the following steps to elicit security requirements:

1. Identify critical assets in the system
2. Define security goals for each asset
3. Identify threats to each of these security goals, by identifying the stakeholders that may want to cause harm to the system
4. Identify and analyze risks for the threats, using risk assessment techniques
5. Define security requirements for the risks.

Although misuse cases have proven to be easy to understand and use in different fields, the identification of threats is limited by the creativity and knowledge of the security experts involved in the brainstorming sessions [35].

2.3.4 OCTAVE

Operationally Critical Threat, Asset and Vulnerability Evaluation (OCTAVE) is a risk-based strategic assessment and planning technique for security [36]. Instead of targeting technological risk and focusing on tactical issues, OCTAVE is targeted on organizational risk and focuses on practice-related issues.

The technique captures the current state of security practice within an organization based on the knowledge of its people regarding the security-related practices and processes. Since it involves people from different parts of the organization, not only from IT departments, the technique can assist an organization to balance the three key aspects: operational risk, security practices, and technology.

The approach is driven by two of these aspects, operational risk and security practices. Technology is only examined to refine the view of the current security practices. OCTAVE is self-directed, meaning that the organization is required to manage the evaluation process and make decisions to address the protection of information [36]. To achieve that, the analysis team, consisting of people both from operational and IT units of the organization, follows an asset-driven evaluation approach which involves the following steps [36]:

1. identify information-related assets (e.g., information and systems) that are important to the organization
2. focus risk analysis activities on those assets judged to be most critical to the organization
3. consider the relationships among critical assets, the threats to those assets, and vulnerabilities (both organizational and technological) that can expose assets to threats
4. evaluate risks in an operational context - how they are used to conduct an organization's business and how those assets are at risk due to security threats
5. create a practice-based protection strategy for organizational improvement as well as risk mitigation plans to reduce the risk to the organization's critical assets

The organizational, technological, and analysis aspects of an information security risk evaluation are complemented by a three-phase approach which enables organizational personnel to assemble a comprehensive picture of the organization's information security needs. An overview of the phases is presented in Figure 2.1.

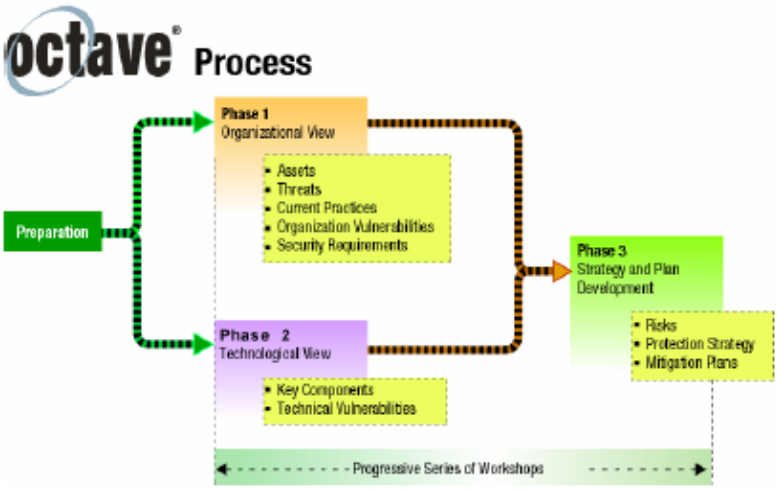


Figure 2.1: OCTAVE Phases

2.3.5 PASTA

Process for Attack Simulation and Threat Analysis (PASTA) is threat modeling methodology introduced by Marco Morana and Tony UV to assist an organization in managing technical and non-technical risks caused by threats seeking to exploit vulnerabilities in the application [37]. The process begins by defining the business objectives, security and compliance requirements, along with a business impact analysis. Similar to other threat modeling techniques, a Data Flow Diagram (DFD) is modeled to identify threats and vulnerabilities using abuse cases, threat trees, and a scoring system [37]. The final step involves calculation of risk and business impact and adaptations of countermeasures to reduce the impact of threats. To summarize, PASTA provides an attacker-centric view of threats in combination with risk and impact analysis that can help organizations establish an asset-centric approach, not only focusing on technical threats [37].

2.3.6 TVRA

Threat Vulnerability and Risk Analysis (TVRA) is an assessment method developed by the European Telecommunication Standards Institute (ETSI) [38], used to identify assets and their associated weaknesses, vulnerabilities with their corresponding threats and their negative impact on the security attributes, namely confidentiality, integrity, availability, authenticity, and accountability (CIAA).

The threats are then classified under one of the following categories: Interception, Manipulation, Denial of service, Repudiation of sending, Repudiation of receiving. A risk value is then calculated depending on the impact and likelihood of the evaluated threats, and a set of countermeasures are derived.

Finally, a cost-benefit analysis is performed to select the most suitable countermeasures to reduce the likelihood of the threat to occur and its potential impact [38].

Integrating a countermeasure adds assets to the Target of Evaluation (TOE) and could introduce new vulnerabilities. Therefore, it's crucial that the TVRA method is performed until all risks are at an acceptable level. An overview of the TVRA process is provided below [38]:

1. Identification of the Target of Evaluation (TOE) resulting in a high level description of the main assets of the TOE and the TOE environment and a specification of the goal, purpose and scope of the TVRA.
2. Identification of the objectives resulting in a high level statement of the security aims and issues to be resolved.
3. Identification of the functional security requirements, derived from the objectives from step 2.
4. Inventory of the assets as refinements of the high level asset descriptions from step 1 and additional assets as a result of steps 2 and 3.
5. Identification and classification of the vulnerabilities in the system, the threats that can exploit them, and the unwanted incidents that may result.
6. Quantifying the occurrence likelihood and impact of the threats.
7. Establishment of the risks.
8. Identification of countermeasures framework (conceptual) resulting in a list of alternative security services and capabilities needed to reduce the risk.
9. Countermeasure cost-benefit analysis (including security requirements cost-benefit analysis depending on the scope and purpose of the TVRA) to identify the best fit security services and capabilities amongst alternatives from step 8.
10. Specification of detailed requirements for the security services and capabilities from step 9.

It is recommended to document the assets, threats, vulnerabilities, weaknesses and countermeasures found during the process in case the analysis needs to be repeated whenever the design or the environment changes [38].

2.3.7 EVITA - THROP

Threat and Operability analysis (THROP) method is part of the E-Safety Vehicle Intrusion Protected Applications (EVITA) project. The method considers potential threats for a particular feature from a functional perspective, using attack trees to define the potential threats regarding the functionality of the analyzed feature. THROP first identifies the primary functions of the feature, then applies guide-words to identify potential threats so that it can determine potential worst-case scenario outcomes from the potential malicious behavior [39].

The risk level determination is adopted from ISO 26262 (ASIL determination) [40] based on a combination of severity, attack probability, and controllability measures, which are mapped to a qualitative risk level (R0 to R7 and R7+) for classification of the security threats. The severity classification separates different aspects of the consequences of security threats (operational, safety, privacy, and financial) [39].

2.3.8 HEAVENS Security Model

The primary objective of the HEAVENS security model is to derive security requirements for the target of evaluation (TOE), similar to the notion of deriving functional safety requirements as described by ISO 26262 [41], [40]. To achieve that, HEAVENS security model analyses threats using STRIDE and then ranks according to their security level (SL), which is based on the threat level (TL) and impact level (IL), in order to derive security requirements. An overview of the HEAVENS security model workflow is presented in Figure 2.2 below.

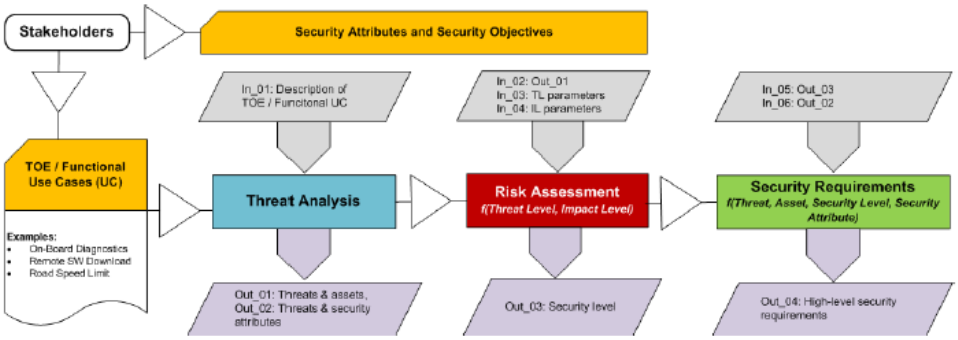


Figure 2.2: Workflow of HEAVENS security model

The threat level (TL) is determined based on the expertise of the attacker, his knowledge of the TOE, the window of opportunity and the equipment required to facilitate the attack. Each of these parameters are assigned a value between 0 and 3, corresponding to none, low, medium and high. The summation of these values is used to define the threat level which can be 0 for none, 1 for low, 2 for medium, 3 for high and 5 for critical [41].

In a similar way, the threat impact level (IL) is derived based on the impact the threat will have on each of the 4 security objectives adopted by HEAVENS, safety, financial, operational, and privacy and legislation. For each of the security objectives mentioned, a value is assigned and then the sum is calculated to derive the IL [41]. Finally the security level (SL) can be derived by combining the TL and IL as shown in Figure 2.3.

Security Level (SL)		Impact Level (IL)				
		0	1	2	3	4
Threat Level (TL)	0	QM	QM	QM	QM	Low
	1	QM	Low	Low	Low	Medium
	2	QM	Low	Medium	Medium	High
	3	QM	Low	Medium	High	High
	4	Low	Medium	High	High	Critical

Figure 2.3: HEAVENS Security Level (SL)

3
Compiling the Catalog

Before compiling the catalog, it was necessary to define the scope of design flaws, examine available resources and databases as well as define a process for the search, filtering and selection of the entries. An overview of the process of compiling the catalog is presented in Figure 3.1. It consists of the following steps:

1. **Define the scope of design flaws.** Flaws related to specific technologies, protocols or implementation level details are not considered relevant for our study. For instance, flaws of TCP/IP protocol or Object Oriented Programming (OOP) design flaws such as god class or feature envy which are not directly related to security are out of scope.
2. **Collect data.** In this step the available resources were reviewed in order to collect the data which will be the base for the catalog. The resources reviewed are the following:

   2.1 Review collections design flaws available in the literature or previous work.

   2.2 Review threat taxonomies for different fields and domains to identify the most common threats. These threats were then filtered, so that only the ones related to design flaws were kept. STRIDE threat trees are also included in this step.

   2.3 Review the entries of CWE, OWASP top 10 and SANS 25 to identify weaknesses, threats and vulnerabilities related to design flaws.
3. **Derive a list of threats and weaknesses** from the data collected in the previous step. This means that the most common entries were grouped and and filtered in order to remove duplicate or similar entries.
4. **Compile a catalog of design flaws.** For each of the entries in the list derived from step 3, examine what are the underlying issues that cause them. For example, STRIDE and several threat taxonomies define spoofing identity as a threat, but this is usually the result of how credentials are managed (storage, transferred through channels) or insufficient authentication mechanisms (in some cases single factor authentication may be insufficient). The catalog of design flaws will be comprised of these causes.
5. **Define detection guidelines** for each of the flaws in the catalog. For each entry, steps were defined to help identify the relevant elements in the architecture along with a set of questions for each of these elements. This way, depending on how these questions are answered, a flaw can be detected.
6. **Present the catalog** that contains design flaws along with guidelines on how to detect them in an architecture. Identifying and addressing each of these flaws can potentially prevent one or more threats.

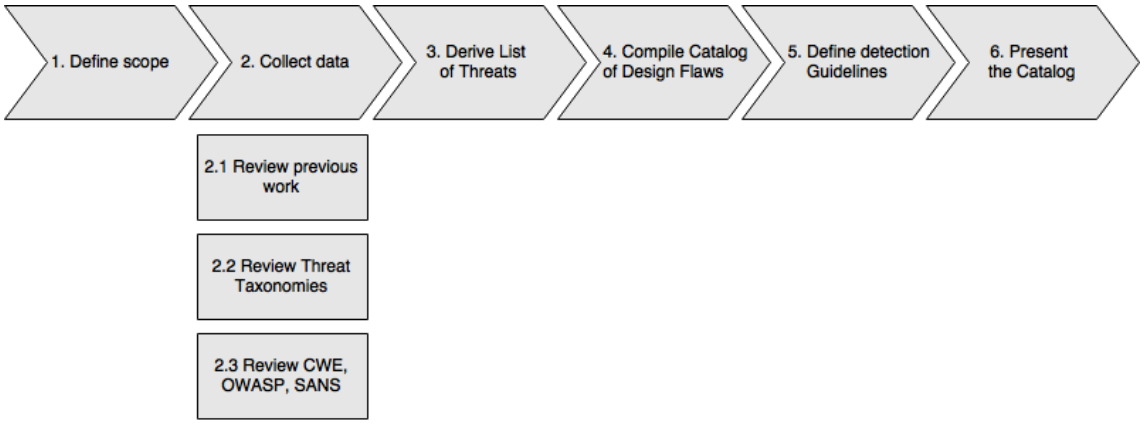


Figure 3.1: Overview of the process to compile the catalog.

3.1 Defining the scope of design flaws

As mentioned in section 2.1.3, there are a number of terms found in the literature when referring to design flaws. Additionally, depending on the study, the design flaws identified can be on different abstraction levels of the architecture. For example, design flaws such as "god class" or "feature envy" can be found in studies on OOP systems. Other studies are focused on identifying flaws in communication protocols or specific technologies. However, since the primary goal of this study is to identify design flaws early on, where the architecture is not very detailed, flaws such as the ones mentioned above are considered out of scope. This also allows for the resulting catalog to contain design flaws that can be present in different types of systems, not restricting it to a single domain.

3.2 Data collection

In order to collect the data, that the catalog is based on, related work regarding catalogs of design flaws, knowledge bases of threats, weaknesses and design or security patterns, as well as threat taxonomies were reviewed. The remaining of this section presents these resources.

3.2.1 Threat and Vulnerability Taxonomies

This section presents taxonomies considered during our literature review and used as a source for data collection phase. The main focus was on studying taxonomies of security threats, however these usually included attacks and vulnerabilities organized under generic threat categories. Additionally, taxonomies of vulnerabilities often include design flaws, so certain vulnerability taxonomies were also considered.

Igure and Williams [42] provide a comprehensive view on taxonomies of attacks and vulnerabilities, in the form of a review covering taxonomies presented from 1974 until 2006. They analyze the effectiveness of these taxonomies for use in security assessment processes. The important properties of the various taxonomies are also

summarized to provide a framework for organizing information regarding known attack and vulnerabilities. Although a large number of the taxonomies reviewed refer to operating system flaws and implementation level vulnerabilities, the study provides a useful overview of the most common security issues occurring in different fields. Finally, they argue that a taxonomy has to be system specific for it be of use [42].

Joshi et al. [43] present a comparison of classification methods for 25 taxonomies of vulnerabilities. Their study covers taxonomies from 1974 until 2014 and the authors point out that no taxonomy can be complete and that it is necessary to update taxonomies in order for them to be useful with respect to security issues associated with today's software products and attack mechanisms [43].

Tripathi and Singh [44] provide an overview of proposed taxonomies of vulnerabilities in literature and point out the need for a single unified taxonomy that will be updated constantly in order for it to be useful. They claim that although there are several taxonomies proposed throughout the years, most of them are now outdated and they follow different classification schemes. Tripathi and Singh conclude that there is a need for standardization regarding information security taxonomies which contribute in improving the security and risk assessment of information systems.

Uzunov and Fernandez [45] propose an extensible two-level pattern-based taxonomy of security threats for distributed systems. Each threat in this taxonomy is encapsulated in an abstract software pattern (threat pattern) that can be specialized for other systems and technologies. This way the taxonomy can be extended through specialization. The first level of the taxonomy organizes threats into 8 classes, which are identity attacks, network communication attacks, network protocol attacks, passing illegal data, stored data attacks, remote information inference, loss of accountability and uncontrolled operations [45]. The second level organizes threat patterns in 4 meta security threat classes, which are cryptographic attacks, countermeasure design, configuration/administration and network protocol attacks. Although not all of the threats can be attributed to design flaws, since the majority of the threats are quite abstract, several of the entries were considered during data collection phase [45].

Babar et al. [46] provide an overview of the objectives in the field of IoT (Internet of Things) and summarize the key properties, challenges and high level security requirements in the field. They also propose a taxonomy of the major security challenges of 6 categories which are identity management, communication threat, storage management, dynamic binding, embedded security and physical threat. Finally, they propose a 3-dimensional security model for IoT (cube structure) to depict the convergence of security, trust and privacy. The taxonomy contains of high level threats such as spoofing or authentication issues which was used as part of the data collected for the creation of our catalog [46].

Silva et al. [47] present a taxonomy of security threats in the web ecosystem di-

vided in 3 domains (service, service consumer and social engineering), each of which contains a subset of the total of 21 attack vectors. These attack vectors can be described as high level threats such as missing authentication and a number of them was included in the initial data collected for the catalog.

Tsipenyuk et al. [48] propose a taxonomy of software security errors which is divided in "7 plus 1" kingdoms. Each kingdom contains a number of coding errors, referred to as phylum. Although the taxonomy contains coding errors, some of them can be attributed to design flaws. Of particular interest is the *"Security features"* kingdom which refers to coding errors regarding least privilege violation, credentials management etc.

3.2.2 Vulnerability Databases

This section provides a short description of the various databases available, that contain information regarding vulnerabilities and weaknesses. These databases were examined during the data collection phase of our study.

Common Vulnerabilities and Exposures (CVE) was launched in 1999 and is the largest and most updated vulnerability database. CVE is maintained by the MITRE corporation and provides a reference-method for publicly known information-security vulnerabilities and exposures. Every vulnerability has a unique ID and most other databases have links to CVE based on those IDs [49].

Common Weakness enumeration (CWE) is a community-developed list of common software security weaknesses. CWE aids developers and security practitioners since it serves as a common language for describing security weaknesses in architecture and implementation. It also provides different mitigation and prevention techniques that could be used to eliminate weaknesses. CWE contains around 1005 weaknesses and each entry is cross referenced with different taxonomies, CVE and attack databases such as CAPEC [29]. Additionally, CWE provides a number of views, which include categorization of the entries based or taxonomies in the literature, research concepts, time of introduction to the system etc.

3.2.3 Other Resources

This section lists resources that are made available by organizations such as OWASP and SANS institute. The resources provided are in the form of top 10/25 lists which are updated every couple of years and provide information regarding the most common vulnerabilities. These resources were also examined during the data collection phase.

The Open Web Application Security Project (OWASP) provides information about risks associated with the most common Web application security flaws and provides recommendations for dealing with those flaws. The OWASP publishes

the flaws in a form of popular Top Ten list every year and was founded 2001 as a not-for-profit charitable organization [50].

SANS top 25 This is a list of the most widespread and critical errors that can lead to serious vulnerabilities in software. The list is the result of collaboration between the SANS Institute, MITRE, and many top software security experts in the US and Europe. It combines knowledge gained during the development of SANS Top 20 attack vectors and MITRE's Common Weakness Enumeration (CWE) [51].

Preliminary List of Vulnerability Examples for Researchers (PLOVER)
PLOVER is a report by Christey PLOVER [52] aimed to assist in the creation of the Common Weakness Enumeration (CWE) initiative. The report provides an informal taxonomy and presents concepts for a general "vulnerability theory", identification of research gaps, discussion of terminology, and a mapping from PLOVER categories to 1500 CVE names, including an index of hard-to-classify examples [52].

STRIDE Threat Trees
STRIDE is currently one of the most commonly used techniques when it comes to threat modeling. Microsoft has also created a tool which is based on STRIDE. Its acronym defines a threat classification which consists of Spoofing, Tampering, Repudiation, Information disclosure, Denial of service and Elevation of privilege. In [32], Adam Shostack presents 15 threats trees where each leaf is a threat that can realize the root node.

3.2.4 Related Work Regarding Catalogs of Design Flaws

Previous work regarding compilation of catalogs of design flaws can be found in [3], [53], and [54]. In [3], Da Silva Santos shifts through the entries of CWE and collects those that refer to weaknesses that are attributed to design flaws. The result is a new catalog, CAWE, which contains 384 architectural weaknesses, however, it doesn't provide any guidelines on how to detect these flaws. At the time our study was conducted, access to this catalog was not possible, so we cannot make any conclusions about the relevance to our work.

Terkelsen [53] follows a similar approach, were a subset of the CVE entries is examined. Then these entries are filtered based on a decision tree, whether they are design flaws or not, and are classified based on a classification scheme. Similar to [3], it provides no guidelines on how to detect these flaws.

The most recent work is presented in [54]. This document was published by IEEE CSD and lists the 10 most common design flaws. The flaws listed there however have a broad scope and can be broken down to a number of erroneous design decisions. Thus, each flaw listed in the document can be considered as a category of flaws.

3.3 Deriving a list of most common threats and weaknesses.

After reviewing the resources described in section 3.2, a number of threats, weaknesses and vulnerabilities were collected from each resource. These were then cross-referenced and merged to eliminate any duplicates and similar entries, resulting in a list that would be used as the base for the catalog. A part of the list derived at this step is presented in Table 3.1 along with related CWE entries.

#	Threats	Related CWE Entries
1	Missing Authentication	287, 306, 862
2	Authentication Bypass using an alternate path	288, 592
3	Authentication Bypass using a spoofed identity	290
4	Authentication Bypass by Capture Replay Methods (Session hijacking)	294, 592, 300
5	Authentication Bypass by Assuming Data Stored on Client are Immutable	302, 471
6	Relying on Single Factor Authentication	308, 654
7	Not Setting appropriate Session Timeout	384, 613
8	Session Hijacking	384
9	Not Re-authenticating Before Critical Operation	306
10	Relying on shared resources for authentication	291, 350
11	Downgrade Authentication (providing multiple authentication mechanisms)	757
12	Weak Change Management	261, 262, 263, 521,522,549
13	Use of Key Distribution Center (KDC)	264
14	Credentials Stored in Client	522
15	Credentials Stored in Server	522
16	Missing Authorization	862
17	Execution of External Programs	829, 830
18	Missing Access Check	638, 284, 285, 272, 280, 269
19	Not Using Complete Mediation	638
20	Not considering Context when authorizing	270
21	Not revoking authorization	299, 672
22	Credentials not Encrypted while Stored or in Transit.	319, 201, 209
23	Sensitive Data not Encrypted in Storage or in Transit	319, 201, 209, 315
24	Communication Channels are Not Properly Encrypted	300, 940, 941, 923
25	Channel Accessible by Non-Endpoint ('Man-in-the-Middle')	300
26	Use of Custom/Weak Encryption Algorithms.	327, 326, 325
27	Absence of well established Trust Boundaries	501, 485, 269
28	Not validating data from Untrusted sources	20
29	Incoming data from untrusted source don't have an upper bound	130
30	Use of untrusted APIs or Libraries	648
31	Insufficient auditing	778
32	Unlimited resource usage	400, 770
33	Acceptance of Extraneous Untrusted Data With Trusted Data	349
34	Clean cached/temp files	377
35	Behavioral change in new versions or environment	439
36	Containment Errors (Container Errors)	216

Table 3.1: Part of threats and weaknesses collected in step 3 and related CWE entries.

3.4 Compile the catalog of design flaws.

Once a list of common threats was created, the next step was to analyze the threats and identify the underlying causes for each one. In some cases the threat was directly connected to a design flaw, such as "Missing Authentication". In other cases, however, the threats can be attributed to one or more design flaws in the architecture. For example, "Authentication Bypass using a spoofed identity" can be caused by a number of factors such as communication through an insecure channel which allows for sniffing packets, along with not proper management credentials or a weak authentication scheme. Another example would be "Session Hijacking" which can be caused by non-secure communication channels, exposure or guessing of session identifier as well as not invalidating a session after logout or timeout.

In order to be able to link the threats and weaknesses to design flaws, CWE was used as a common ground. CWE provides mapping to threat taxonomies in the literature and is also mapped to CVE and CAPEC databases as well as OWASP and other online resources[1]. It also provides a categorization of the weaknesses in a class hierarchy and provides several views of the enumeration based on different research or development concepts. Furthermore, for each weakness CWE provides, among other things, a description, potential mitigations and relationships to other entries.
For each threat or weakness, in order to identify the underlying cause, the following steps were taken:

1. **Examine the description of the threat/weakness.**
2. **Identify the related CWE entry.** This was mostly for threats, since the weaknesses were derived from CWE already. For threats that only offered related CVE, CAPEC or OWASP top ten links, these were used to trace the CWE entry. In the rare case that a threat was not linked to any of these resources, based on the description of the threat and any provided examples, it would be mapped to an attack pattern from CAPEC and then to CWE or to a CWE entry directly if possible. Finally, since threat taxonomies aim for completeness, they usually contain entries such as “other threat” or abstract threats regarding future extensions of the system and considerations about the users. These are usually difficult to define though, so they were not considered.
3. **Check for parent classes in CWE.** Once all the threats and weaknesses were connected to CWE entries, the next step was to examine the parent classes of these entries. This allowed to move to a higher layer of abstraction, since some of these entries were for specific technologies. At this point, all the CWE entries are introduced during “design and architecture” phase, so they can be considered as design flaws.
4. **Group the entries.** Since many entries were regarding the same concepts, these were grouped as a single entry in the catalog of design flaws, since it would allow an analyst to consider this concept for the overall system. For example, entry 4. Insufficient Session Management is the result of flaws related to the session ID management, session timeout, session fixation etc. However,

[1]Some of these mappings can be found here : `https://cwe.mitre.org/data/index.html`

it is easier to consider all these issues altogether when examining the sessions in an architecture, rather than having multiple entries regarding this concept.

5. **Compile the catalog.** This is the final step in this process which resulted in the entries of catalog as presented in Table 3.2 below.

#	Design Flaw
1	Missing authentication
2	Authentication Bypass using an alternate path
3	Relying on Single Factor Authentication
4	Insufficient Session Management
5	Downgrade Authentication
6	Insufficient Cryptographic keys management
7	Missing Authorization
8	Unmonitored Execution of External Applications
9	Missing Access Control
10	Not Re-authenticating Before Critical Operation
11	Not considering Context when authorizing
12	Not revoking authorization
13	Insecure data storage
14	Insufficient credentials management
15	Insecure data exposure
16	Use of Custom/Weak Encryption Algorithms.
17	Not validating input/data
18	Insufficient auditing
19	Uncontrolled Resource Consumption

Table 3.2: Design flaws to be included in the catalog.

Examining the entries in Table 3.2 one can notice that certain entries may appear to be less specific. For example, the entry referring to cryptographic key management involves the generation, distribution, storage, usage as well as destruction of the keys in the system. As already mentioned, although these could be documented as separate entries, it may be easier for a practitioner to consider the overall lifecycle of keys in the architecture, rather that going through multiple entries. Instead detection guidelines in this entry are defined for each part of the lifecycle.

3.5 Defining detection guidelines.

To detect these design flaws, it is important to determine the elements in the architecture that are relevant for each flaw. For example, for the entry "Insecure data storage", if the type, importance/sensitivity level, location and allowed access is determined, then a set of questions can help identify any possible issues.

In order to define these guidelines, a number of resources were reviewed, such as OWASP, CWE, NIST special publications and security checklists. As mentioned in the previous section, all the entries of the catalog are linked to CWE, CVE and

CAPEC entries. All these databases provide suggestions for potential mitigation for each entry. These can be either referring to certain design patterns, design principles or suggestions for the design based on best practices. If some of these were related to a specific technology they were filtered out.

Furthermore, OWASP provides guidelines for several concepts in the form of "Cheat-sheets" [2], to assist practitioners to secure systems against the most common security threats, as those are listed in the OWASP top 10 projects. In a similar way, NIST provides a series of special publications, which provide guidelines, recommendations and reference material for practitioners[3].

These resources were examined in order to form the questions in the quidelines for each entry. First, a checklist for each entry was formed, based on the suggestions for potential mitigation in CWE, which was then extended to include guidelines and recommendations from OWASP and NIST resources. The result of this process was the final catalog of design flaws, where each entry has a description and detection section. During this process, there was continuous feedback from our academic supervisors at Chalmers, as well as our industrial supervisors at Volvo. The full catalog is included in Appendix A.

[2]OWASP Cheat Sheet Series: `https://www.owasp.org/index.php/OWASP_Cheat_Sheet_Series`

[3]NIST special publications series: `http://csrc.nist.gov/publications/PubsSPs.html`

4
Evaluation of the Catalog

This section describes the evaluation process for the catalog of design flaws proposed in this paper. To gather the necessary data to answer the research questions, a descriptive study was done to evaluate the use of the catalog.

According to Grimes and Schulz [55], descriptive studies are "*concerned with and designed only to describe the existing distribution of variables, without regard to causal or other hypotheses.*" Since the catalog differentiates from existing approaches in the literature, this study focuses more on how the catalog can perform rather than comparing it to other approaches. Additionally, a descriptive study allows for observations to be made on how the approach can be improved. Alternatively, controlled experiments could be conducted instead. These focus more on specific variables, measures and how these relate [56]. However, this would require the formulation of hypotheses, which would be difficult in this case, since there were no expectations regarding the effectiveness and productivity of the approach.

The evaluation took place in two different environments. The catalog was evaluated by students from Chalmers University of Technology, in the form of a task as this is described in section 4.2.3, as well as in a professional environment with two practitioners at Volvo Group Trucks Technology, using an architecture provided by them. Prior to the evaluation, a pilot test was done with a student to assess the overall process, get an indication of the time required to complete the task and obtain feedback on the catalog. After completion of the task, the participants were asked to fill in a short questionnaire, containing questions regarding the process.

4.1 Preparation for the evaluation

This section describes the steps taken to prepare for the evaluation of the approach. This includes the selection of measurements, a pilot run to get initial feedback on the task, the catalog and an estimation for the time required to complete the task, as well as how a baseline was established in order to process the participants answers in order to obtain the necessary measurements.

4.1.1 Selected measurements and metrics

The first research question (RQ1) examines the effectiveness of the catalog, which can be described in terms of precision (P) and recall (R). These can be determined

by measuring the number of correct (*true positives*), incorrect (*false positives*) and overlooked (*false negatives*) design flaws detected using the approach.

The second research question (RQ2) examines the productivity of the analyst using the approach. This can be determined by measuring the overall time and number of correct design flaws *(true positives)* per unit of time (hour).

The third research question (RQ3) examines how this approach can complement existing knowledge-based threat modeling techniques. This can be determined by analyzing the design flaws identified by the participants and determine if these flaws could be the potential cause of one or more threats. By examining this, a conclusion can be drawn to see if identifying design flaws and addressing them early on can potentially mitigate threats discovered using a threat modeling technique.

In order for the research questions of the study to be answered, the following measurements were selected.

- Time: The time required to complete the task, excluding breaks.
- True Positives (TP): These are the flaws detected by the participants that are correct flaws and are also included in the baseline.
- False Positives (FP): These are the flaws detected by the participants that are incorrect.
- False Negatives (FN): These are the flaws overlooked by the participants, meaning flaws that are included in the baseline but not detected by the participants.
- Precision (P): The percentage of detected flaws that are correct. This can be calculated by using the following formula: P = TP/(TP+FP)
- Recall (R): The percentage of the existing flaws, based on the baseline, that are detected. This can be calculated by using the following formula: R= TP/(TP+FN).
- Productivity: The amount of true positives found per unit of time (hour).

4.1.2 Pilot run

Prior to the evaluation, a pilot run took place with a student to get an initial assessment of the process as well as feedback. The student was given the task and documents that would be used for the students. Although these results are not taken into consideration for this study, the authors received valuable feedback regarding both the catalog and the process. Before conducting the pilot run, some of the questions in the detection guidelines couldn't be answered with a yes or no. This way the participant would have to consider the answers given for the entry and decide whether or not it is a flaw. This also allowed for some things to not be answered clearly. As the student that did the pilot run pointed out, this could easily lead to frustration and perhaps make the process harder and more time consuming.

This lead to the restructuring of some of the questions in the catalog, so that all the questions now would be answered with yes/no and no would suggest the presence

of this flaw. The pilot run also provided an estimation of what would be the time required to complete the task. This helped planning the meetings with the rest of the participants and informing them about the estimated duration of the task so that any implications would be avoided.

4.2 Evaluation with students

4.2.1 Experimental object

The Home Monitoring System – HomeSys was selected as an experimental object. The system was introduced during the DAT220 Advanced Software Architecture course in the Software Engineering MSc program at Chalmers University of Technology. Homesys is a remote platform for monitoring and managing of residential apartments and houses. The platform is a cloud-based system that prompts its customer with notification during critical events such as an intrusion, fire, water leak or any other notification that can be configured by the customer.

The participants were provided with the documentation of the system, consisting of the following documents:

- **Home Monitoring System (HomeSys) - Requirements**
  This document provides the functional and non-functional requirements for the system in the form of use cases.
- **Home Monitoring System (HomeSys) - Description of the problem domain**
  Contains the description of the system, the problem space, the stakeholders involved and the additional constraints in the domain.
- **Home Monitoring System (HomeSys) - Architectural description**
  Contains a context diagram, component diagrams for the main components of the system along with description of the functionality of their sub-components and a deployment diagram. The architecture has been designed based on the domain analysis, the requirements and the priorities of these requirements.

4.2.2 Participants

A total of 4 students participated in the study. The participants were selected among the students that had successfully completed the course. This means that they were already familiar with the system used for the study, as well as basic security concepts, design patterns and threat modeling techniques (STRIDE) that were introduced during the course. In spite of that, the participants were given the documentation of the system a week before, so that they would have the time to read the documents and prepare for the task.

4.2.3 Task

Each participant was given a set of documents including the system's documentation, the catalog of design flaws and a form to document their findings. The

participants were asked to go through each entry in the provided catalog, use the guidelines to detect flaws in the system's architecture and document every instance of each flaw they find properly.

For each entry in the catalog, the participants would have to identify certain elements in the architecture and then for each of these elements answer a series of questions. Answering one or more of these questions with a No would suggest that this particular flaw is likely to be present in the architecture. The participants were encouraged to read through the entirety of the provided documentation to collect the necessary information to answer these questions.

In case the information required to follow the guidelines was not provided in the documentation, or was not clearly specified, they participants were instructed to consider this as a flaw. If they thought a particular entry was not applicable in the system under examination due to some restrictions or regulations in the domain, they were instructed to mark that entry as not applicable. They were allowed to use any part of the documentation and create additional UML diagrams if they found it necessary, as long as they would provide those along with their answers.

The instructions for the task were given to the participants in text, in the form of a task description, and then repeated by the authors. They were also informed that during the tasks, the authors would not answer any questions regarding the content of the catalog or the guidelines. The only questions that were answered were regarding the procedure. The participants were also informed that there is no time limit and were encouraged to be as thorough in their analysis as possible. Since the task was time consuming, they were encouraged to take short breaks, but asked not to discuss the results.

4.2.4 Establishing a Baseline

To be able to assess the answers given by the participants, a baseline was created. The baseline represents the idealistic result obtained by thoroughly examining the system, using the catalog of design flaws. In order to create the baseline, each of the authors did an initial analysis of the system separately and then compared the results. In most cases the two analyses were in agreement. For any differences, there was a discussion regarding the assumptions made and the reasons why it is considered a flaw or not, until a consensus was reached. The outcome of this process was a list of 49 design flaws identified in HomeSys architecture and is presented in Table 4.1 below.

#	Flaw Instance	CatalogEntry
1	Mote with MicroPnP component	1. Missing authentication
2	Gateway connecting to Cloud	1. Missing authentication
3	Gateway Mgmt interacting with GWFacade	1. Missing authentication
4	GWFacade interacting with DBMonitor	1. Missing authentication
5	NotificationHandler with DBMonitor	1. Missing authentication
6	NotificationController with CustomerDB	1. Missing authentication
7	Customer Mgmt with DBMonitor	1. Missing authentication
8	Operator Mgmt with DBMonitor	1. Missing authentication
9	Operator Mgmt with CustomerDB	1. Missing authentication
10	Customer client to Customer Façade	2. Authentication Bypass using alternate path
11	Operator Client to Operator Façade	2. Authentication Bypass using alternate path
12	Gateway to Cloud	2. Authentication Bypass using alternate path
13	Mote to MicroPnP	2. Authentication Bypass using alternate path
14	Password for customer/operator	3. Relying on single factor Authentication
15	Customer client to Customer Façade	4. Insufficient Session Management
16	Operator Client to Operator Façade	4. Insufficient Session Management
17	Keys management not specified.	6.Insufficient Cryptographic Key Management
18	Customer - no authorization	7. Missing Authorization
19	Operator - no authorization	7. Missing Authorization
20	Gateway - no authorization	7. Missing Authorization
21	Mote - no authorization	7. Missing Authorization
22	Access control mechanism not specified	8. Missing Access Control
23	UC4 Associate GW to Customer - no reauthentication	9. Not reauthenticating before critical operation
24	UC9 Rules Reconfiguration - no reauthentication	9. Not reauthenticating before critical operation
25	UC14 Changing SLA- no reauthentication	9. Not reauthenticating before critical operation
26	UC15 Unregister Customer- no reauthentication	9. Not reauthenticating before critical operation
27	Customer when unregistering - authorization not revoked	12. Not revoking authorization
28	Gateway when customer unregisters - authorization not revoked	12. Not revoking authorization
29	Operator when leaving company- authorization not revoked	12. Not revoking authorization
30	Customer Mgmt for configuring rules, SLA change, Deactivating account	12. Not revoking authorization
31	SensorData in MainDB	13. Insecure Data Storage
32	SensorData in GWDB	13. Insecure Data Storage
33	Profile info in Customer/Operator DB	13. Insecure Data Storage
34	Logs in LogDB	13. Insecure Data Storage
35	Customer/Operator credentials storage, encryption, change mgmt	14. Insufficient Credentials management
36	Sensor Data from Mote to GW	15. Insecure data exposure
37	Sensor Data/Alarms from GW to GWFacade	15. Insecure data exposure
38	Sensor Data/Alarms from GWFacade to MainDB	15. Insecure data exposure
39	Sensor Data, Customer Info, View Logs from Customer/Operator Facade to Clients	15. Insecure data exposure
40	No encryption algorithms specified	16. Use of Custom/Weak Encryption Algorithms
41	Mote data used for alarms/events and stored in Main DB not validated	17. Not validating input/data
42	Gateway facade does not validate data from GWs	17. Not validating input/data
43	Logs have no back-up	18. Insufficient auditing
44	Motes to Gateway channel - no monitor, throttling or notification	19. Uncontrolled Resource Consumption
45	Gateway to Cloud channel - no monitor, throttling or notification	19. Uncontrolled Resource Consumption
46	NotificationController Channels - no monitor, throttling or notification	19. Uncontrolled Resource Consumption
47	CustomerClient to Customer Facade channel - no monitor, throttling or notification	19. Uncontrolled Resource Consumption
48	OperatorClient to Operator Facade channel - no monitor, throttling or notification	19. Uncontrolled Resource Consumption
49	GW DB capacity - no monitor or notification, possible to deplete	19. Uncontrolled Resource Consumption

Table 4.1: Baseline of flaws for HomeSys.

4.3 Relating design flaws to threats

After the completion of the evaluation with the students and the processing of the results, the next step was to associate the design flaws found with the threats derived from STRIDE analysis. The STRIDE analysis was performed by 4 groups of student within the scope of Advanced Software Architecture course. It was ensured that these students did not overlap with the participants for the study. The results were processed and provided by our academic supervisors. These results were then analyzed in regards to how they can be linked to the design flaws detected by the catalog, as described in section 5.2.

4.4 Evaluation at Volvo Group Truck Technologies

A similar task was given to practitioners at Volvo Group Truck Technologies (GTT). Two practitioners were made available to the authors to complete a similar task, as

the one that was given to the students, but on an architecture provided by Volvo, in the form of an ECU topology. The practitioners performed the task as a pair, since one of them was software oriented and the other towards mechanical engineering. Due to the size of the system, initially the goal was to select a smaller part of the architecture to analyze. However after consideration, the participants decided to use the whole system architecture, since this way they would be able to cover the entirety of the catalog in their analysis.

5
Results

This section presents the data collected from the evaluation process and describes how these were processed in order to calculate the measurements required to answer the research questions.

5.1 Results from students

Once the students completed the task, the forms they were given to document the flaws were collected. Along with the documentation of the system as well as the copy of the catalog were also collected. Although the students were specifically asked to document each flaw properly, in some cases there were references to figures in the architecture or the checkboxes for the questions in the catalog.

Gathering all these information was helpful in understanding the reasoning behind each flaw they documented. In cases where the answer was unclear or the student answered one of the questions without defining a specific element, this was considered as a false negative (overlooked flaw), while if the answer pointed towards a flaw that wasn't present in the baseline, it was considered a false positive (incorrect flaw). After carefully considering each answer, the correct, incorrect and overlooked flaws were counted and the precision and recall were calculated for each student as well as the mean and standard deviation for the sample. A summary of the results is presented in Table 5.1 below.

Student	**TP**	**FP**	**FN**	**Precision**	**Recall**	**Time (mins)**	**Productivity (TP/h)**
1	25	2	24	0.93	0.51	146	10.27
2	28	3	21	0.90	0.57	166	10.12
3	9	4	40	0.69	0.18	201	2.69
4	20	4	29	0.83	0.41	191	6.28
mean	20.5	3.25	28.5	0.84	0.42	176	7.34
s	8.35	0.96	8.35	0.11	0.17	24.83	3.61

Table 5.1: Student results and measurements

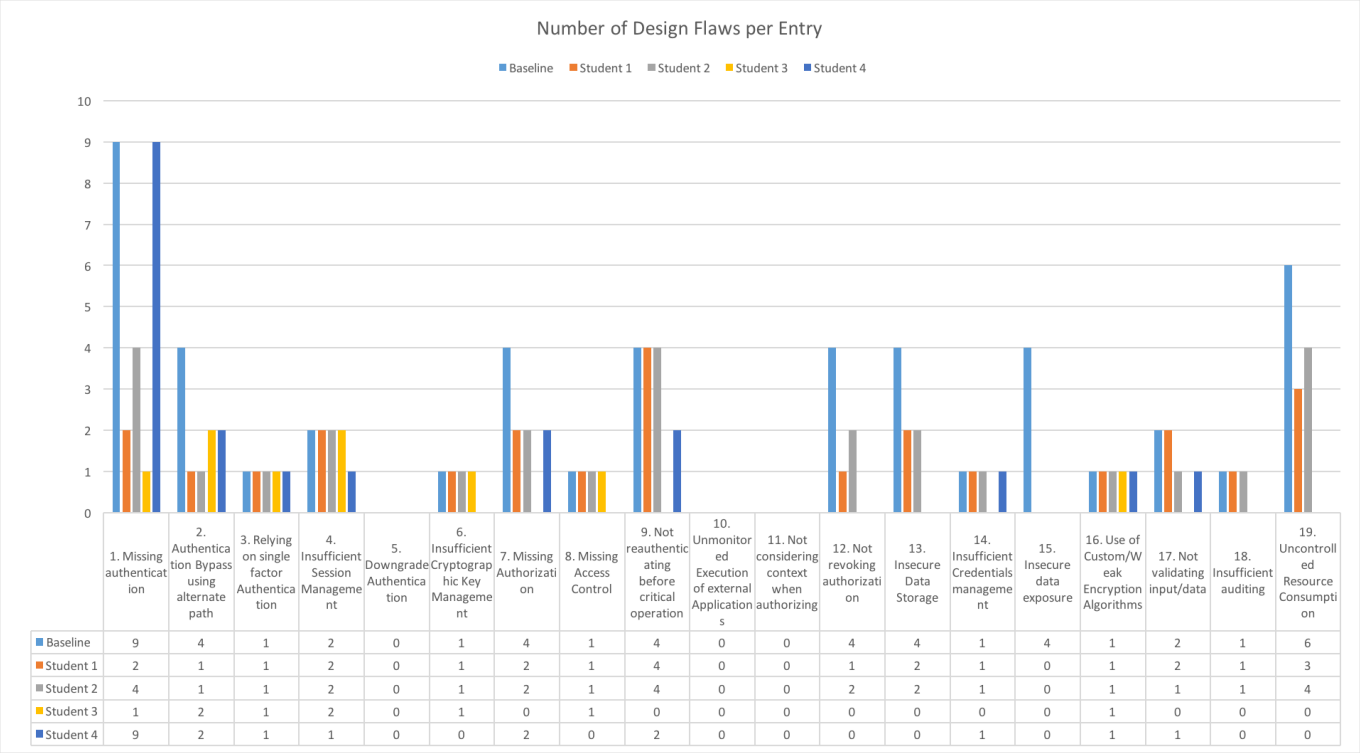
According to the data presented in Table 5.1, the number of correct design flaws (TP) detected by the students on average is 20.5 with a standard deviation of 8.35, while for incorrect design flaws (FP) the average is 3.25 with a standard deviation of 0.96. The average precision (P) of the students is 0.84 with a standard deviation

of 0.11.

The average of overlooked design flaws (FN) is 28.75 with a standard deviation of 8.38, while average recall (R) is 0.42 with a standard deviation of 0.17.

Finally, the average time spent by the students to complete the task is 176 mins with a standard deviation of 24.83 while the average productivity of the students is 7.34 correct flaws per hour (TP/Hour).

Although the precision of the approach is high, meaning that most of the flaws detected were correct, recall average of 42% suggest that there was a substantial number of flaws overlooked. The chart in Figure 5.1 provides an overview of the number of flaws detected for each entry as well as how many each student detected.



	1. Missing authentication	2. Authentication Bypass using alternate path	3. Relying on single factor Authentication	4. Insufficient Session Management	5. Downgrade Authentication	6. Insufficient Cryptographic Key Management	7. Missing Authorization	8. Missing Access Control	9. Not reauthenticating before critical operation	10. Unmonitored Execution of external Applications	11. Not considering context when authorizing	12. Not revoking authorization	13. Insecure Data Storage	14. Insufficient Credentials management	15. Insecure data exposure	16. Use of Custom/Weak Encryption Algorithms	17. Not validating input/data	18. Insufficient auditing	19. Uncontrolled Resource Consumption
Baseline	9	4	1	2	0	1	4	1	4	0	0	4	4	1	4	1	2	1	6
Student 1	2	1	1	2	0	1	2	1	4	0	0	1	2	1	0	1	2	1	3
Student 2	4	1	1	2	0	1	2	1	4	0	0	2	2	1	0	1	1	1	4
Student 3	1	2	1	2	0	1	0	1	0	0	0	0	0	0	0	1	0	0	0
Student 4	9	2	1	1	0	0	2	0	2	0	0	0	0	1	0	1	1	0	0

Figure 5.1: Number of flaws detected in HomeSys per catalog entry.

Based on the data presented in Figure 5.1 it appears that the highest number of overlooked flaws is for entries 1, 7, 12, 13, 15 and 19 in the catalog. Considering the elements in the architecture the students were required to identify for those entries as well as the specific flaws they overlooked, the following observations can be made:

- 1. Missing authentication: Most of the overlooked flaws were regarding processes not authenticating with Database component, which is in a different part of the system.
- 7. Missing authorization: Based on the flaws identified by the students, most of the overlooked flaws are regarding the part of the system concerning the Gateway and the Motes. In other words, students mostly focused on HomeSys

Cloud and considered authorization mostly for users (customers and operators).
- 12. Not revoking authorization: Similarly to the first entry, students didn't consider the processes.
- 13. Insecure data storage: Students didn't consider the local DB in gateway, even though it stores sensitive data and can be considered as a high-value asset.
- 15. Insecure data exposure: Although they made the assumption that no channels are secure in the system, in most cases students didn't identify the communication channels through which sensitive data would be transferred.
- 19. Uncontrolled resource consumption: Another case where the main focus was on HomeSys Cloud. The students didn't regard the channels and DB in the Gateway as critical resources, even though they are.

Based on observations made during the task, the students mostly focused on the components diagrams provided in the documentation, while a substantial amount of information is provided in the rest of the documents as well. Also there was no indication that the students identified the security objectives of the system in order to determine the high-value assets or the critical operations regarding those.

Furthermore, a substantial amount of the overlooked flaws were regarding processes. These would potentially be easier to detect in a behavioral model rather than structural models, as the ones provided in the documentation of HomeSys, since it would allow for the participants to have a better view of the interactions between the different components.

Finally, based on the data collected by the questionnaire, although it appears that the documentation provided sufficient information to complete the task, the respondents would prefer to have a model organizing all this information better. This could potentially make it easier to collect the required information to detect a flaw and improve the recall rate, as well as the overall productivity.

5.2 Relating design flaws to threats derived from STRIDE

In order to relate the design flaws to threats, the design flaws detected using the catalog were checked against the threats identified using STRIDE. Among the threat modeling techniques, STRIDE was chosen for the reasons listed below:
- STRIDE is one of the most widely used threat modeling techniques.
- The participants in our study were familiar with the technique, as it was introduced in the Advanced Software Architecture course.
- Results from STRIDE analysis from 4 different groups of students on HomeSys were available to us.

The analysis of each group contained a Data Flow diagram (DFD), along with a list of the correct threats each group identified. Since there were many similarities

between the threats listed among the 4 groups, these were merged in a single list and the duplicates removed. During this process, special attention was given to the different DFDs that were created from the groups, to ensure that the similar threats were indeed duplicates. In one case, a threat listed by a group, although it was correct based on the DFD, they hadn't modeled the accountability component in the DFD. This meant that the threat would not be possible if the architecture and the DFD were in agreement. This resulted in this threat being removed from the list. This resulted in a list of 34 unique threats.

ID	ThreatName	Related Flaw ID in Baseline
1	Corrupt Mote	-
2	Spoofing mote	1, 13
3	Overflowing the gateway with requests	36, 44
4	Spoofing the gateway	2, 12
5	Disable the watchdog	-
6	Disable the periodic scheduler	-
7	Tampering with GW DB	-
8	False rules	-
9	Spoofing the DB Monitor	-
10	DoS on Primary DB	-
11	Accessing sensitive info from DBs	-
12	Tampering with DS (enabled by spoofing the monitorDB)	-
13	Spoofing the customer	10, 14, 15, 35, 39
14	Spoofing the operator	11, 14, 16, 35, 39
15	Spying on customers	11, 14, 16, 35, 39
16	Customer elevating privilages	18, 22, 25, 33
17	Generating login requests	15, 16, 39, 47, 48
18	Sniffing the credentials	10, 11, 15, 16, 35, 39
19	SQL injection on client side	-
20	Disable logging operator actions	-
21	Generating view sensor data requests	-
22	Tampering with customer data	-
23	Fill up the Event/Alarm queue	-
24	Fill up the sensor queue	-
25	Sniff the sensor data	37, 38
26	Sniff events	37, 38
27	Generate gateway connections	45
28	Attacking logs	43
29	Tamper with sensor data before it reaches cloud	37
30	Tamper with configuration in transit	37
31	Spoofing the cloud	2, 3, 12
32	Reveil sensor data or tamper with rules	-
33	Block/redirect data to cloud	37
34	Blocking notification channels	46

Table 5.2: Threats from STRIDE analysis and related flaws in the baseline

Once a list of unique threats was established, each threat was checked against the design flaws detected by using the catalog, as these are listed in the baseline. In order to relate each threat to detected design flaws, each threat was analyzed to identify which elements of the DFD it involved, what were the preconditions for the threat to be applicable and how this could be realized. Then each flaw was

considered to decide whether addressing this flaw would mitigate the threat.

As an example, one of the threats in the list is "Spoofing Mote". According to the description provided for the threat, it is possible to connect to the gateway posing as a Mote by sniffing its IP. This is possible since there is no authentication between the Mote and the MicroPnP component in the Gateway. Additionally, according to STRIDE threat tree regarding spoofing an entity, one of the leafs of the tree in "No Authentication". The baseline (Table 4.1), contains two flaws regarding the lack of authentication between motes and gateways (flaws number 1 and 13 in the baseline) as shown in Table 5.2. Similar process was followed for the rest of the threats. The analysis of all 34 threats is presented in Appendix C.

While some threats can only be associated with one flaw, others can be linked to multiple. Table 5.3 provides an overview of the number of associated flaws each student found for each threat.

ID	ThreatName	Baseline	Student 1	Student 2	Student 3	Student 4
1	Corrupt Mote	-	-	-	-	-
2	Spoofing mote	2	2	2	2	2
3	Overflowing the gateway with requests	2	-	1	-	-
4	Spoofing the gateway	2	1	-	1	2
5	Disable the watchdog	-	-	-	-	-
6	Disable the periodic scheduler	-	-	-	-	-
7	Tampering with GW DB	-	-	-	-	-
8	False rules	-	-	-	-	-
9	Spoofing the DB Monitor	-	-	-	-	-
10	DoS on Primary DB	-	-	-	-	-
11	Accessing sensitive info from DBs	-	-	-	-	-
12	Tampering with DS (enabled by spoofing the monitorDB)	-	-	-	-	-
13	Spoofing the customer	5	3	3	2	3
14	Spoofing the operator	5	3	3	2	2
15	Spying on customers	5	3	3	2	2
16	Customer elevating privilages	4	4	4	1	2
17	Generating login requests	5	4	4	2	1
18	Sniffing the credentials	5	3	3	2	2
19	SQL injection on client side	-	-	-	-	-
20	Disable logging operator actions	-	-	-	-	-
21	Generating view sensor data requests	-	-	-	-	-
22	Tampering with customer data	-	-	-	-	-
23	Fill up the Event/Alarm queue	-	-	-	-	-
24	Fill up the sensor queue	-	-	-	-	-
25	Sniff the sensor data	2	-	-	-	-
26	Sniff events	2	-	-	-	-
27	Generate gateway connections	1	1	1	-	-
28	Attacking logs	1	1	1	-	-
29	Tamper with sensor data before it reaches cloud	1	-	-	-	-
30	Tamper with configuration in transit	1	-	-	-	-
31	Spoofing the cloud	3	1	-	1	3
32	Reveil sensor data or tamper with rules	-	-	-	-	-
33	Block/redirect data to cloud	1	-	-	-	-
34	Blocking notification channels	1	-	-	-	-

Table 5.3: Number of design flaws in baseline and student results associated with each threat

Overall, the design flaws listed in the baseline were associated with 18 out of 34 threats from STRIDE. Table 5.4 presents an overview of the number of threats that could be linked to the baseline and the student results. For the threats that a design flaw could not be associated, this was in most cases due to the prerequisites of the

threats or their close relation to implementation issues. For example, the threat regarding DoS on primary DB has as a prerequisite the spoofing of DBMonitor, which is the mechanism supposed to manage and monitor the DB according to the documentation. While the catalog can detect flaws regarding resource monitoring and consumption, this is based on the existence of a monitoring mechanism. If this mechanism is compromised, then there is no way to prevent DoS.

Similarly, several threats require an Elevation of Privilege (EoP) attack to occur in order to be applicable, especially regarding the operator. However, in the documentation it is mentioned that the operator has access to the whole system. So in order to detect flaws regarding the authorization levels and access control it would require to make assumption about the system and authorization policies, which would be out of scope. Finally, for threats that require physical access to compromise a device, there is little to be done from an architecture perspective.

	Baseline	Student 1	Student 2	Student 3	Student 4
Number of threats	18	11	10	9	9
(%)	53	32.4	29.4	26.5	26.5

Table 5.4: Number of threats for which associated flaws were found.

As Table 5.4 shows, the design flaws listed in the baseline could be associated with about 53% of the overall threats identified using STRIDE. Also, it is important to note that even though the recall rate of the students ranged from 18% to 57% the number of threats associated to the flaws they found ranges from 9 to 11. This is due to the fact that a threat can be linked to multiple design flaws and a design flaw can contribute to a number of threats, which is why it is not clear whether these threats could be eliminated or just mitigated by addressing the design flaws.

Another aspect that can be examined is the time required to prepare for using the catalog to detect flaws. Unlike several threat modeling techniques, using the catalog does not require any expertise or creation of specific UML diagrams, such as DFDs or elaborate models, such the ones described in [7] and [8]. However, the observations made during the study, as well as the answers of the participants in the questionnaire, suggest that the effectiveness of the approach could benefit from a more organized documentation or model to be used as input.

5.3 Results from Volvo

Initially the goal was to obtain similar measurements as with the students, however due to the fact that the architecture of the system was not made available to the authors and could not be made public, a baseline could not be established in order to compare the results. Additionally, since the project was at early stages, security was not yet considered for the system. Thus, most of the flaws in the results were due to the fact that a lot of things were not specified yet in the architecture. The

answers provided by the participants are presented in Table 5.5 below.

Design Flaw	Description
1. Missing Authentication	Authentication not yet considered
2. Authentication Bypass using alternate path	Authentication not yet considered. No Encryption
3. Relying on Single Factor Authentication	Authentication Scheme not yet determined
4. Insufficient Session Management	Session management unknown
5. Downgrade Authentication	Authentication scheme not decided
6. Insufficient Cryptographic keys management	Key management not yet decided
7. Missing Authorization	Privileges not used. No policy for entire platform
8. Missing Access Control	No access control policy
9. Not Re-authenticating Before Critical Operation	No re-authentication policy decided
10. Unmonitored Execution of External Applications	We will have mechanisms in place e.g. Hypervisor
11. Not considering Context when authorizing	Not applicable yet considered
12. Not revoking authorization	Not applicable yet considered
13. Insecure data storage	Will be considered
14. Insufficient credentials management	No credentials management policy yet
15. Insecure data exposure	Not considered for external entities
16. Use of Custom/Weak Encryption Algorithms	Not considered yet
17. Not validating input/data	Payload data is to be validated by application
18. Insufficient auditing	No auditing mechanism yet
19. Uncontrolled Resource Consumption	Will be considered

Table 5.5: Results from Volvo.

Looking at the answers provided by the participants it is hard to draw any concrete conclusions regarding the flaws detected, since they do not refer to specific elements in the architecture. However, as one of the participants commented on the questionnaire, the catalog could also be used as a way to ensure that important aspects have been taken into consideration in the architecture. In other words, the catalog could also be used as a form of checklist during the design phase.

Additionally, based on the answers of both practitioners in the questionnaire, using a model describing the software components and their interactions would be useful in this case. This is in agreement with the observations we made based on the results from the students. Many of the overlooked flaws in that case were related to authentication of processes and revoking authorization. This is usually easier to detect in a behavioral diagram.

Another interesting observation was that during the process the participants tried to classify the flaws according to their impact on the system. However, to the best of our knowledge, there is no risk assessment methodology for design flaws at the time this study was conducted.

6
Discussion

This section discusses the results of the study and how the research questions can be answered, comparison to similar approaches, discussion regarding the threats to validity for the study and future work.

6.1 Effectiveness of the catalog

Using the data collected from the students, we can answer the first research question regarding the effectiveness of the approach (RQ1). The precision of the students participated in our study, ranged from 0.69 to 0.93 with an average of 0.84, which means that for every 10 flaws a participant detected, 8 were correct flaws. The recall rate of the students using the catalog ranged from 0.18 to 0.57 with an average of 0.42. This means that on average the participants found about 42% of the overall flaws in the system.
It is also important to consider that the participants were not familiar with the catalog before the task and that the information required to detect the flaws was not organized in a single document. Providing the required information in a more structured way and giving the participants an introduction to the approach could potentially increase the recall rate. Finally, based on the nature of the overlooked flaws, it appears that the use of behavioral models, apart from the structural models provided in the documentation, could potentially decrease the amount of overlooked flaws.

Compared to similar approaches, the tool proposed by Almorsy et al. in [8] has a precision rate of 90% and a recall rate of 89%. However, their approach is based on an automated tool, while ours is manual. Furthermore, they only look for 4 specific types of attacks in the system, namely MITM, DoS, Data Tampering and Injection attacks.

6.2 Productivity of the analyst using the catalog

Regarding the productivity using this approach (RQ2), the average time required for the participants to complete the process was about 2 hours and 46 minutes. Their productivity ranged from 2.69 to 10.27 with an average of 7.34 correct flaws identified per hour (TP/Hour). We notice that the deviation in the productivity as well as the recall rate is quite high between the participants. This could be attributed to a number of factors, including the type of models provided in the documentation,

the familiarity of the participants with the system and its domain, as well as their experience as analysts. However, based on the data collected during the study we cannot make any conclusive arguments regarding these factors.

6.3 How the catalog can complement threat modeling techniques

Regarding how the catalog can complement existing knowledge-based threat modeling techniques (RQ3), certain things need to be considered. Our data suggest that the flaws detected using the catalog can be associated with threats derived from STRIDE analysis. However, we cannot make a conclusive argument on whether these threats would be eliminated or just mitigated to a certain degree. This prevents us from making a conclusion on whether identifying the flaws related to these threats would reduce the time and effort required for threat modeling.

To have conclusive results about this, the detected flaws associated with these threats should be fixed in the architecture and STRIDE applied again in order to compare the results. However, due to time restrictions, this was not possible during our study.

However, even if a threat is not eliminated by addressing the related flaws, it can be mitigated to a certain degree. Identifying design flaws earlier in the development process is usually less costly to fix. This could potentially reduce the overall cost related to threat mitigation.

Additionally, since the approach doesn't require any specific models or detail level for the architecture, it can be used at different abstraction levels in different stages of the design phase as a form of a checklist for issues to consider which can have an impact on the security of the system. This, however, would probably require the catalog and its guidelines to be refactored for use at different abstraction levels.

6.4 Using the catalog in the automotive industry context

Although we were not able to gather any concrete data, regarding the effectiveness and productivity of the catalog, in the study conducted with practitioners at Volvo, some interesting observations were made. To begin with, the practitioners mentioned that in order for the catalog to be more effective, it should be more domain specific. Since the catalog is based on most common threats collected from different fields, some of its entries or guidelines may need to be adapted to a specific domain in order to improve its effectiveness. Additionally, based on their comments, apart from the ECU topology used, behavioral models describing the interactions between the different components would be helpful for the analysis.

Of particular interest was the observation that the practitioners were trying to assess the impact of the design flaw while using the catalog and in some cases disregard a flaw as non-important. Although most threat modeling technique involves risk assessment for the threats found, we are not aware of any methodology for risk assessment regarding design flaws.

6.5 Similar Studies

Scandariato, Wuyts and Joosen conducted a study in [6], where they evaluate STRIDE in terms of productivity, correctness and completeness. The study involved ten teams of students, with an average size of 4 members per team, with a similar background to the ones that participated in our study. According to their results, the average productivity of the students is 1.8 threats per hour, the average precision is 0.81 and the average recall rate is 0.36 for a system similar in complexity to the one we used for the study, but bigger in size.

We cannot make a direct comparison of the results from our study to the ones from [6], since STRIDE identifies threats while our approach aims in detecting flaws and as mentioned before, the connection between these two concepts is not very clear. However, we can get an indication of what precision and recall rates are in other manual approaches.

6.6 Threats to Validity

This section presents threats that may have an impact in the validity of the study. These can be classified as construct, internal and external validity threats. Construct validity refers to the measurements selected for the study and how well these measure the effect we want to measure. Internal validity focuses on how certain we can be that the treatment caused the outcome. In other words, how sure we can be that the outcome wasn't affected by factors not controlled in the study. Finally, external validity refers to whether the results can be generalized outside the scope of the study [57].

6.6.1 Construct Validity

Regarding construct validity, a possible threat would be the metrics selected for the first and second research question, namely precision, recall and productivity. However, these measurements were selected based on the fact that most studies we found in the literature examining similar approaches use them. This would also allow for comparison of the findings of the study to other similar studies.

Another possible threat to construct validity could be the procedures followed to obtain the measurements. Specifically, in order to measure the time required by the participants to compete the task in our study, both authors measured it separately and then compared the results. In most cases the differences were minor, about

1 minute, which is negligible compared to the average time taken to complete the task.

6.6.2 Internal Validity

Regarding the internal validity of the study, the process followed to compile the catalog of design flaws as well as the guidelines could be considered a threat. In order to mitigate this, the authors tried to connect all the threats to CWE which can be considered a credible source, in order to provide a way to trace the steps taken to derive the design flaws from threats. Similarly, for the guidelines for each entry, resources such as CWE, CAPEC, OWASP and NIST were used. Finally, there was regular feedback from both academic and industrial supervisors.

Since the task to be performed required a substantial amount of time, participants could become exhausted in the process. To mitigate that threat, we informed the participants beforehand, giving them an estimation of the time required to complete the task, based on the pilot run. In addition, the participants were encouraged to regularly take short breaks but asked not to discuss the result among each other. Regardless, there was no case of 2 students taking a break at the same time.

Another threat to the internal validity was the student's limited understanding and familiarity of the Home Monitoring system. To mitigate this, the student were given documentation of the system a week before the evaluation, and they were also reminded to read the documents two days prior to the task.

During the task, in order to avoid introducing bias in the process, the participants were informed that any questions regarding the contents of the catalog would not be answered. The only questions answered, were regarding the process.

In order to mitigate any threats to validity regarding the baseline, the authors did a separate analysis and then compared the results. For any differences, a discussion was made in order to reach a final decision. A similar process was followed for the students results. Ideally, these results would be also examined by an independent party, but due to time restrictions, this was not possible.

The threats from STRIDE analysis done by the groups of students within the scope of DAT220 Advanced Software Architecture course were processed and provided to us by a PhD student.

An additional threat to the internal validity could be that the authors mistakenly merged different threats as duplicates during the processing of the STRIDE results. However, both authors did an individual processing of the STRIDE results and compared their results to detect any uncommon results.

Finally, during the analysis of each threat from STRIDE in order to relate it to design flaws, there could be some mistakes. Although the analysis was repeated

twice and there were minor differences, this could also be a threat to the validity of the study.

6.6.3 External Validity

Regarding external validity, the applicability of catalog in different domains could be considered a threat. However, during the process of compiling the catalog, we considered a number of threat taxonomies and weaknesses from various fields. And since the design flaws defined in our catalog are at an abstraction level where specific technologies, implementation and protocols details are not of relevance, we believe that the approach could be applicable in different domains.

Due to the nature of the task being time consuming, finding participants willing to spend a few hours on a task was rather difficult. Therefore a convenience sample of students was selected among classmates in the Software Engineering MSc program in Chalmers University of Technology. Due to the limited time, randomization of the sample was not possible. However, we made sure that the students selected had no prior knowledge regarding our work and fulfilled the criteria of having completed the DAT220 Advanced Software Architecture course. This, along with the fact that the overall sample size is small and not necessarily representative of the population, makes it difficult to generalize the results of our study.

Finally, the system used for the evaluation task (HomeSys) has a relatively small size and complexity. Using our approach in systems larger in size and complexity could yield different results. However, doing that would require a significant amount of time and was not possible in our study.

6.7 Future Work

The results of our study show that using our approach, design flaws related to security threats can be identified in early phases. However we cannot provide a conclusive answer on whether these threats are eliminated or not. Further work needs to be done on this part, to investigate whether addressing these flaws in the architecture can reduce the amount of threats identified later on using threat modeling techniques.

In addition, the catalog proposed in the study can be further extended and refined to cover more threats from different domains. It would also be interesting to examine how easily it can be adapted for use in specific domains.

Another interesting point to consider for future work would be to investigate how the impact of such design flaws can be determined. Although there are several risk assessment methodologies regarding threats, to the best of our knowledge, there is no methodology for risk assessment regarding design flaws.

7
Conclusion

This study proposes a manual approach, in the form of a catalog of design flaws along with detection guidelines, to aid practitioners in the identification of architectural design flaws that can lead to security threats. Compared to existing research approaches and current threat modeling techniques, our approach doesn't require elaborate models, implementation details or particular expertise to use. The approach is aimed to be used by practitioners while designing the system, utilizing existing models, without requiring a deep security knowledge. The catalog of design flaws was compiled by examining threat taxonomies and weaknesses in different domains, and filtering out the ones related to design flaws. In order to simplify the detection of these flaws, guidelines following an algorithmic approach were defined for each entry in the catalog.

To evaluate the effectiveness and productivity of the approach, a descriptive study was conducted involving students examining a small sized system. The identified flaws were then compared against threats derived from the same system using STRIDE, to investigate how the flaws detected can be related to these threats. Overall, the participants using approach had an average precision of 0.84 while the average recall rate was 0.42, with the average productivity of the participants reaching 7.34 correct flaws per hour (TP/h). Based on the observations made throughout the study, the precision, recall and productivity appears to be influenced by a number of factors such as the type of models used for the analysis (structural or behavioral), the experience of the analysts as well as their familiarity with the system under analysis and the catalog.

The catalog was also tested by practitioners from Volvo Group Trucks Technology (GTT) to investigate how it can perform in an industrial setting. Although the data collected during the process were not sufficient to make any conclusions regarding the effectiveness and productivity of the catalog in an industrial context, valuable observations were made regarding the applicability of the approach in different domains as well as how this can be further extended in order to be used as a checklist for security aspects to consider at different abstraction levels.

In addition, the study examines how the approach can complement current threat modeling activities taking place in early phases of the development process. Due to time limitations, the exact nature of the relation between the threats and the detected flaws could not be determined. This study, however, can serve as a base for further research on the topic.

Additional points to consider for future work include an investigation of the relation of design flaws with security threats, whether these can be eliminated by addressing these flaws and how this can affect the time and effort required for threat modeling activities.

Finally, the impact these flaws can have on the security properties of a system, to the best of our knowledge, cannot be captured by any of the currently used risk assessment methodologies.

References

[1] Gary McGraw. *Software security: building security in*, volume 1. Addison-Wesley Professional, 2006.

[2] Brian Chess and Gary McGraw. Static analysis for security. *IEEE Security & Privacy*, 2(6):76–79, 2004.

[3] Joanna Cecilia da Silva Santos. Toward establishing a catalog of security architecture weaknesses. 2016.

[4] Mehdi Mirakhorli, Yonghee Shin, Jane Cleland-Huang, and Murat Cinar. A tactic-centric approach for automating traceability of quality concerns. In *Software Engineering (ICSE), 2012 34th International Conference on*, pages 639–649. IEEE, 2012.

[5] Jilles Van Gurp, Sjaak Brinkkemper, and Jan Bosch. Design preservation over subsequent releases of a software product: a case study of baan erp. *Journal of Software Maintenance and Evolution: Research and Practice*, 17(4):277–306, 2005.

[6] Riccardo Scandariato, Kim Wuyts, and Wouter Joosen. A descriptive study of microsoft's threat modeling technique. *Requirements Engineering*, 20(2):163–180, 2015.

[7] Bernhard J Berger, Karsten Sohr, and Rainer Koschke. Automatically extracting threats from extended data flow diagrams. In *International Symposium on Engineering Secure Software and Systems*, pages 56–71. Springer, 2016.

[8] Mohamed Almorsy, John Grundy, and Amani S Ibrahim. Automated software architecture security risk analysis using formalized signatures. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 662–671. IEEE Press, 2013.

[9] Holisec security proposal ver. 2.0,.

[10] Jerome H Saltzer and Michael D Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

[11] Mark Graff and Kenneth R Van Wyk. *Secure coding: principles and practices*. " O'Reilly Media, Inc.", 2003.

[12] Ronald Wassermann and Betty HC Cheng. Security patterns. In *Michigan State University, PLoP Conf.* Citeseer, 2003.

[13] Markus Schumacher and Utz Roedig. *Security engineering with patterns: origins, theoretical models, and new applications*, volume 2754. Springer, 2003.

[14] It security pattern. `http://www.opensecurityarchitecture.org/cms/definitions/security_patterns`. (Accessed on 04/04/2017).

[15] Joseph Yoder and Jeffrey Barcalow. Architectural patterns for enabling application security. *Urbana*, 51:61801, 1998.

[16] Priya Anand, Jungwoo Ryoo, and Rick Kazman. Vulnerability-based security pattern categorization in search of missing patterns. In *Availability, Reliability and Security (ARES), 2014 Ninth International Conference on*, pages 476–483. IEEE, 2014.

[17] Aleem Khalid Alvi and Mohammad Zulkernine. A natural classification scheme for software security patterns. In *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on*, pages 113–120. IEEE, 2011.

[18] Munawar Hafiz, Paul Adamczyk, and Ralph E Johnson. Growing a pattern language (for security). In *Proceedings of the ACM international symposium on New ideas, new paradigms, and reflections on programming and software*, pages 139–158. ACM, 2012.

[19] Aleem Khalid Alvi and Mohammad Zulkernine. A comparative study of software security pattern classifications. In *Availability, Reliability and Security (ARES), 2012 Seventh International Conference on*, pages 582–589. IEEE, 2012.

[20] Darrell M Kienzle, Matthew C Elder, David Tyree, and James Edwards-Hewitt. Security patterns repository version 1.0. *DARPA, Washington DC*, 2002.

[21] Joshua Garcia, Daniel Popescu, George Edwards, and Nenad Medvidovic. Toward a catalogue of architectural bad smells. In *International Conference on the Quality of Software Architectures*, pages 146–162. Springer, 2009.

[22] Naouel Moha, Duc-loc Huynh, Yann-Gaël Guéhéneuc, and Ptidej Team. A taxonomy and a first study of design pattern defects. *STEP 2005*, page 225, 2005.

[23] Ieee software blog: Common architecture weakness enumeration (cawe). `http://blog.ieeesoftware.org/2016/04/common-architecture-weakness.html`. (Accessed on 04/05/2017).

[24] Hugo Sica de Andrade, Eduardo Almeida, and Ivica Crnkovic. Architectural bad smells in software product lines: An exploratory study. In *Proceedings of the WICSA 2014 Companion Volume*, page 12. ACM, 2014.

[25] Dianxiang Xu and Joshua Pauli. Threat-driven design and analysis of secure software architectures. *Journal of Information Assurance and Security*, 1(3):171–180, 2006.

[26] Adam Shostack. Experiences threat modeling at microsoft. In *Modeling Security Workshop. Dept. of Computing, Lancaster University, UK*, 2008.

[27] Marco D'Ambros, Alberto Bacchelli, and Michele Lanza. On the impact of design flaws on software defects. In *Quality Software (QSIC), 2010 10th International Conference on*, pages 23–31. IEEE, 2010.

[28] Capec - common attack pattern enumeration and classification (capec). `https://capec.mitre.org/`. (Accessed on 04/04/2017).

[29] Cwe - common weakness enumeration. `https://cwe.mitre.org/`. (Accessed on 04/04/2017).

[30] Michaela Bunke and Karsten Sohr. An architecture-centric approach to detecting security patterns in software. In *International Symposium on Engineering Secure Software and Systems*, pages 156–166. Springer, 2011.

[31] Iso/iec pdts 19249 - catalogue of architectural and design principles for secure products, systems, and applications. https://www.iso.org/standard/64140.html. (Accessed on 05/02/2017).

[32] Adam Shostack. *Threat modeling: Designing for security*. John Wiley & Sons, 2014.

[33] E:_trike_trike_v1_methodology_document-draft.dvi. http://www.octotrike.org/papers/Trike_v1_Methodology_Document-draft.pdf. (Accessed on 05/02/2017).

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

[35] Raimundas Matulevicius, Nicolas Mayer, and Patrick Heymans. Alignment of misuse cases with security risk management. In *Availability, Reliability and Security, 2008. ARES 08. Third International Conference on*, pages 1397–1404. IEEE, 2008.

[36] Christopher Alberts, Audrey Dorofee, James Stevens, and Carol Woody. Introduction to the octave approach. *Pittsburgh, PA, Carnegie Mellon University*, 2003.

[37] Application threat modeling. http://www.isaca.org/chapters5/Ireland/Documents/2013%20Presentations/PASTA%20Methodology%20Appendix%20-%20November%202013.pdf. (Accessed on 05/02/2017).

[38] TS ETSI. 102 165-1:" telecommunications and internet converged services and protocols for advanced networking (tispan). *Methods and protocols*, pages 2011–03.

[39] Georg Macher, Eric Armengaud, Eugen Brenner, and Christian Kreiner. A review of threat analysis and risk assessment methods in the automotive context. In *International Conference on Computer Safety, Reliability, and Security*, pages 130–141. Springer, 2016.

[40] Iso26262 - road vehicles — functional safety. https://www.iso.org/standard/43464.html, Nov 2011. (Accessed on 04/04/2017).

[41] M. Islam and A. Lautenbach. Heavens deliverable d2.0 - security models, ver. 2.0, Mar 2016.

[42] Vinay M Igure and Ronald D Williams. Taxonomies of attacks and vulnerabilities in computer systems. *IEEE Communications Surveys & Tutorials*, 10(1), 2008.

[43] Chanchala Joshi, Umesh Kumar Singh, and Kapil Tarey. A review on taxonomies of attacks and vulnerability in computer and network system. *International Journal*, 5(1), 2015.

[44] Anshu Tripathi and Umesh Kumar Singh. Towards standardization of vulnerability taxonomy. In *Computer Technology and Development (ICCTD), 2010 2nd International Conference on*, pages 379–384. IEEE, 2010.

[45] Anton V Uzunov and Eduardo B Fernandez. An extensible pattern-based library and taxonomy of security threats for distributed systems. *Computer Standards & Interfaces*, 36(4):734–747, 2014.

[46] Sachin Babar, Parikshit Mahalle, Antonietta Stango, Neeli Prasad, and Ramjee Prasad. Proposed security model and threat taxonomy for the internet of things

(iot). In *International Conference on Network Security and Applications*, pages 420–429. Springer, 2010.

[47] Carlo Silva, Ricardo Batista, Ruy Queiroz, Vinicius Garcia, Jose Silva, Daniel Gatti, Rodrigo Assad, Leandro Nascimento, Kellyton Brito, and Pericles Miranda. Towards a taxonomy for security threats on the web ecosystem. In *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP*, pages 584–590. IEEE, 2016.

[48] Katrina Tsipenyuk, Brian Chess, and Gary McGraw. Seven pernicious kingdoms: A taxonomy of software security errors. *IEEE Security & Privacy*, 3(6):81–84, 2005.

[49] Cve - common vulnerabilities and exposures (cve). `https://cve.mitre.org/`. (Accessed on 04/04/2017).

[50] Category:owasp top ten project - owasp. `https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project`. (Accessed on 04/04/2017).

[51] Sans institute - cis critical security controls. `https://www.sans.org/critical-security-controls/`. (Accessed on 04/04/2017).

[52] Steve Christey. Plover: Preliminary list of vulnerability examples for researchers. In *NIST Workshop Defining the State of the Art of Software Security Tools*, 2005.

[53] Harald Terkelsen. Data collection on security flaws caused by design errors. Master's thesis, 2006.

[54] Iván Arce, Kathleen Clark-Fisher, Neil Daswani, Jim DelGrosso, Danny Dhillon, Christoph Kern, Tadayoshi Kohno, Carl Landwehr, Gary McGraw, Brook Schoenfield, et al. Avoiding the top 10 software security design flaws. *Technical report, IEEE Computer Societys Center for Secure Design (CSD)*, 2014.

[55] David A Grimes and Kenneth F Schulz. Descriptive studies: what they can and cannot do. *The Lancet*, 359(9301):145–149, 2002.

[56] Victor R Basili. The role of controlled experiments in software engineering research. In *Empirical Software Engineering Issues. Critical Assessment and Future Directions*, pages 33–37. Springer, 2007.

[57] Robert Feldt and Ana Magazinius. Validity threats in empirical software engineering research-an initial survey. In *SEKE*, pages 374–379, 2010.

A

Catalog of Design Flaws

1. **Missing authentication**

   Description
   This refers to the absence of an authentication mechanism in the system. Apart from external entities, like users or other systems the system may interact with, authentication may be necessary within the system between processes/components/datastores that are located in different trust boundaries.

   Detection
   - Consider the external entities(users/subsystems) that interact with the system and which assets of the system they can access.
   - Determine the processes that interact with high-value assets in the system.
   - For each interaction examine:
     - ☐ If it is an entity: Does the entity go through an authentication point in order to access the asset?
     - ☐ If it is a process: Is the identity of a process accessing datastores or processes in a different part of the system (trust boundaries – requires different privilege levels) verified?

2. **Authentication Bypass using an alternate path**

   Description
   This refers to the case where although there is an authentication mechanism in place, it does not cover all possible entry points to the system. This can be due to the fact that there is a remote access point to the system aiming towards support or maintenance, a possible backdoor. Additionally, a system may make a call to invoke functionality of an external application. In such case, the external application can have access to resources and data of the system if not contained properly.
   Detection
   - Determine the entry points to the system.
   - For each entry point examine:
     - ☐ Does it go through an authentication point?
     - ☐ What kind of assets are accessible through this path? Are their security objectives still achieved?
     - ☐ Is the system protected against MITM and session hijacking attacks? Are the communication channels used encrypted?
     - ☐ Does the system invoke functionality from third party applications? Are these applications subject to proper access control (regarding the resources and data they have access to)?

3. **Relying on Single Factor Authentication**

   Description
   Although single factor authentication schemes, like password, can be efficient and sufficient for many systems, using a multifactor authentication scheme is recommended if the system handles sensitive data or operations, like bank transactions. Even if a strong password management policy is in place, this can be obtained by malicious parties in several ways. For example, many users are used to typing their password whenever they are prompted, so they might accidentally input their password in a form induced by the attacker. Additionally, if the authentication is based on shared resources, like IP address or MAC, these can be easily spoofed or altered leaving the system vulnerable. Such resources can be effective if they are used as part of an authentication scheme, but not as the sole factor for authentication.

Detection

- Determine what factors are used for the authentication scheme.
- For each factor consider:
  - ☐ Does this factor provide sufficient level of security? Is it hard to brute-force, guess or forge it?
  - ☐ Are there any known issues regarding this authentication scheme? If yes, are these properly handled?
  - ☐ Is the use of shared resources like IP address as a sole factor for authentication avoided?

    If it is based on password:
    - ☐ Is there a strong password management policy enforced?
    - ☐ Is the password change management secure? How can the user change/reset credentials. Is he/she notified in such case?

4. **Insufficient Session Management**

Description

Not managing a session properly throughout its lifecycle can leave the system vulnerable to session hijacking attacks. Session management involves creation (the session should be established through a secure channel and session identifier should be encrypted), the time frame the session is active (an attacker might attempt to reuse the session ID to gain access) and its destruction/invalidation (proper session invalidation should take place when the user logs out or session timeout. Not terminating sessions can also lead to resource depletion).

Detection

- Determine which sessions are established in the system and between which endpoints.
- For each session examine:
  - ☐ Is the session established through a secure channel?
  - ☐ Is the session ID encrypted when in transit?
  - ☐ Is the session ID hard to guess?
  - ☐ Is the use of session ID as a parameter in URLs prevented?
  - ☐ Is the session ID validated on server side?
  - ☐ Are secure cookies used?
  - ☐ Is the session ID tied to other user properties like IP, SSL session ID?
  - ☐ Can the same session be accessed simultaneously from two endpoints? Should it?
  - ☐ Is session timeout set? Is it the minimum possible value?
  - ☐ Is the session invalidated on logout?
  - ☐ Is the session ID renewed in the event of privilege change?
  - ☐ Is there a mechanism to monitor the creation/destruction and attempts to connect to a session?
  - ☐ Is the user required to re-authenticate after a period of inactivity?

5. **Downgrade Authentication (providing multiple authentication mechanisms)**

   Description
   The system provides a number of authentication schemes for the user to choose. This may also be connected to different versions of the software/technologies or protocols (i.e. HTTP supports Basic, Digest as well as more complex schemes such as Kerberos, NTLM and OAuth). In order to support older versions where a different authentication scheme was used, the system provides multiple authentication options. This, however, means that the attacker can choose the most vulnerable authentication scheme to attack and at the same time increases the complexity of the system.
   Detection
   - Determine which authentication schemes are available to the users and why.
   - For each authentication scheme examine:
     - ☐ Is the presence of this authentication scheme necessary?
     - ☐ Does it provide sufficient level of security?
     - ☐ Can it be used by all the different type of users? Should it?

6. **Insufficient Cryptographic keys management**

   Description
   This refers to the generation, distribution and storage of the cryptographic keys in the system. A compromise of a key means that every piece of information encrypted with this key is compromised. If there is no mechanism to replace a key or any auditing to aid in recovering, this can lead to serious compromise of the system.

   Detection
   - Determine where cryptographic keys are created, stored, used and how they are distributed.
   - Identify the mechanisms in place to replace a key.
   - Track where the keys are used, for which purpose and who has access to them.
   - For each key examine:
     - ☐ Is it generated within a cryptographic module isolated from the rest of the system?
     - ☐ Does the Random Number Generator (RNG) comply with latest standards?
     - ☐ Is the time the key is in plaintext format minimized?
     - ☐ Is the access to it during that time restricted only to authorized parties?
     - ☐ Are the keys distributed through secure channels?
     - ☐ Is the key stored securely?

- ☐ If the key is stored locally in devices/clients, is it encrypted with Key Encryption Key (KEKs)?
- ☐ Is the key integrity ensured?
- ☐ Is the key used only by processes within the cryptographic module?
- ☐ Is there a secure backup of the datastore that stores the keys?
- ☐ Is all access to the key in plaintext format logged?
- ☐ Is a key only used for a single purpose? (For example only for encrypting data but not other keys)
- ☐ Is the key destroyed after it is no longer needed?
- ☐ Is there a mechanism in place to renew/replace the keys in case they are compromised?

7. **Missing Authorization**

Description
This refers to the absence of an authorization mechanism. This results to users or processes having permissions to access data or functionality in the system that is not intended.

Detection
- Determine the different stakeholders that interact with the system (users, admins etc.).
- Determine the external entities that interact with the system (subsystems, sensors, gateways etc.).
- For these examine:
  - ☐ Is there a policy for assigning privileges? For example policies based on roles (RBAC), assets (MAC), users (DAC).
  - ☐ Is there a mechanism to authorize the actors based on that policy?

8. **Missing Access Control**

Description
User might be properly authenticated and authorized and there is a policy regarding the privileges each users has, but there is no mechanism to enforce this policy. This means that a user can log-in and have access to all the functionality of the system, while he is not supposed to.

Detection
- Determine how access control to assets or functionality is enforced.
- Identify the different access paths throughout the system.
  - ☐ Is there a mechanism in place to check if user has privileges before accessing an asset or process?
  - ☐ Are all data elements or access paths protected by this mechanism?

9. **Not Reauthenticating Before Critical Operation**

   Description
   For certain operations, like online bank transactions or or changing password, or when accessing a protected resource the system should ask the user to re-authenticate. This is important since after the initial authentication of the user, the session could be hijacked or the client might be left unattended. Additionally, critical operations should require elevated privileges which should be assigned during that operation and revoked right after its completion.
   Detection
   - Determine the operations that involve high-value assets.
   - Identify the confidentiality objectives regarding those operations.
   - For each operation consider the following:
     - ☐ Does the system require re-authentication before a critical operation occurs?
     - ☐ Is re-authentication required when there is a change in privilege level?
     - ☐ Is re-authentication required when sensitive data are accessed or altered?

10. **Unmonitored Execution of External Applications**

    Description
    Execution of external applications can result in a compromise of the system if not handled properly. This can be due to abusing the privileges of the caller, not limiting its access to data and resources and can serve as a way to bypass authentication.

    Detection
    - Determine the components or processes that make calls to external applications.
    - Consider the privilege level of these components and what data or functionality is accessible by them.
    - For each case examine:
      - ☐ Does the external application have the least privileges required to perform the operation?
      - ☐ Is its access to data and system resources limited by access control mechanisms?
      - ☐ Is the access to system resources like CPU or memory usage monitored?
      - ☐ Is the access to system resources or data provided through explicit interfaces?
      - ☐ Can it only invoke functionality required for the operation it was called?

11. **Not considering Context when authorizing**

Description
Considering environment, time and other factors when authorizing. Although a user/process might have permissions to do a specific task, these depend on certain conditions like time, location or other factors. For example during an intrusion on the system or heavy load, should a user be authorized to invoke certain functionality?
Detection

- Determine for which assets or processes additional conditions need to be fulfilled to be accessed.
- Consider in which situations normal permissions and access control should not apply.
- For each case examine:
  - ☐ Are the additional conditions, like location or time, considered before authorization is granted?
  - ☐ Consider the circumstances under which authorization should not be granted. Are these considered by the authorization policy?(like during an intrusion, or a critical state of the system)

12. **Not revoking authorization**

Description
In certain cases an application may require elevated privileges to perform a certain operation. After this operation is completed the elevated privileges need to be revoked. The same applies with account termination when for example an employee leaves a company.

Detection

- Determine in which cases authorization needs to be revoked, for example when a user account needs to be deactivated.
- Determine which external entities require elevated privileges.
- Identify the processes that require elevated privileges to perform a critical operation.
- For each external entity (users, subsystems etc) consider:
  - ☐ What operation does the user need the elevated privileges for? Are these revoked after completion of this operation?
  - ☐ Are permissions revoked after account termination?
- For each process examine:
  - ☐ Are any external processes/API called during the time the privilege level is elevated? Do these require the elevated privileges?
  - ☐ Are these privileges revoked after the completion of the action?

13. **Insecure data storage**

Description
Data stored on system are not sufficiently protected. This involves data stored in a resource, which could be a database, memory, temporary files, cookies etc., that are stored in cleartext format or can be accessed by users or processes that should not have permissions to do so. Additionally, sensitive data may be stored along with the rest of the data in one place.
Detection

- Determine the sensitive data stored in the system.
- Locate where these data are stored.
- Determine the actors and processes that can access them.
- For each case examine:
  - ☐ Are these data deleted when they are no longer needed?
  - ☐ Are these data encrypted while stored?
  - ☐ Is there an access control mechanism in place to prevent unauthorised access?
  - ☐ Is any data that come from computations or processing of sensitive data protected as well?
  - ☐ Is access to data logged? (actor, timestamp etc)
  - ☐ Is there a backup for sensitive data?
  - ☐ Are the backups stored in different physical devices?
  - ☐ Is data separated according to their sensitivity level?

14. **Insufficient credentials management**

Description
This entry refers to the storage, transfer and creation/change management of credentials. If the credentials are not encrypted while stored they can easily be stolen, or sniffed if they are sent over an insecure channel. Additionally, if the policy in place to generate the credentials (such as strong passwords) or to reset/change them is not secure, it can lead to a compromise of the system and leave it vulnerable to spoofing threats.

Detection

- Determine where the credentials are stored in the system.
- Track how these are transmitted (through which channels) throughout the system.
- For each case examine:
  - ☐ Are the credentials encrypted in rest?
  - ☐ Are they encrypted in transit?
  - ☐ Is access to the credentials in cleartext logged?
  - ☐ Is there a secure mechanism in place to change or reset credentials?
  - ☐ Is any change in credentials logged?
  - ☐ Is the user notified if a change in the credentials occurs?

15. **Insecure data exposure**

Description
Data is not transferred in a secure way. For example web application uses HTTP instead of HTTPS. This leaves the channel vulnerable to eavesdropping, Man In The Middle (MITM) attacks etc.

Detection
- Locate the valuable information in the model.
- Track them through the architecture to determine where and how they are transferred.
- At each step examine the following:
  - ☐ Is the reuse of packets prevented (Replay attacks)?
  - ☐ Is there any form of timestamping, message sequencing or checksum in the exchanged packages?
  - ☐ Is the traffic over an encrypted channel (SSL/TLS)?

16. **Use of Custom/Weak Encryption Algorithms**

Description
Using a custom or weak cryptographic algorithm can lead to a serious compromise of the system. Even if it is not easy to decrypt the data, information can be inferred by analysing certain patterns like packet transmission, metadata for connection or processing time. Additionally, although an algorithm may be well vetted, it may not be suitable for the purpose it is used or the key size does not provide sufficient degree of security.

Detection
- Determine the cryptographic algorithms used in the system.
- Identify for which purposes they are used.
- For each case examine:
  - ☐ Is the algorithm suitable for the encryption of that particular resource?
  - ☐ Is the key size sufficient?
  - ☐ Are the keys used, managed properly?

17. **Not validating input/data**

Description
Not properly validating data that are received from external sources, can lead to a number of issues, including injection attacks. In certain cases trusted data may be mixed with untrusted data as well. Even data originating from trusted sources might have to be validated, such as session identifiers or protocol headers.

Detection

- Identify the external sources that the system receives data from.
- Locate where this data is used or stored.
- For each case examine:
  - ☐ Is there is a validation step in between.
  - ☐ Is there sufficient validation for the data, for example protocol headers etc?

18. **Insufficient auditing**

Description
Access to critical resources or operations is not logged. This can lead to repudiation issues and possibly makes it more difficult to recover after an attack.

Detection
- Determine for which events or operations it is important to log information in the system
- For each case examine:
  - ☐ Is the access to sensitive data and operations logged?
  - ☐ Are changes in privileges logged?
  - ☐ Are failed attempts to authenticate or access a resource logged?
  - ☐ Do the logs keep information about time, IP, ID when an actor performs an operation?
  - ☐ Is the information logged sufficient to trace where/when an issue occurred and what was altered/corrupted?
  - ☐ Are these logs stored securely?
  - ☐ Is there any backup of the logs in case they are erased by the attacker?

19. **Uncontrolled Resource Consumption**

Description
This entry refers to not properly controlling the use of resources within the system. Not limiting cpu usage, storage, memory, sessions etc can lead to exhaustion of critical resources to the system which can result in denial of service. An increase in workload can also lead to failure and performance degradation when its processing thresholds are exceeded, therefore it's important to balance and limit the resources.

Detection
- Determine which are the critical resources in the system.
- Analyse how each one can be exhausted.
- Consider what assets or operations are accessible to unauthenticated users. For example a site can allow an unauthorised user to browse parts of it.
- For each of these resources examine:
  - ☐ Is there a mechanism to monitor these resources?

- ☐ If a resource reaches a critical threshold, does the system have a throttling mechanism?
- ☐ If a critical resource is depleted, is the system placed in a safe state?
- ☐ Are the appropriate parties notified?

B

Results from students

#	Flaw Instance	CatalogEntry	Student 1	Student 2	Student 3	Student 4
		1. Missing authentication		1 False Positive		
1	Mote with MicroPnP component	1. Missing authentication	Found	Found	Found	Found
2	Gateway connecting to Cloud	1. Missing authentication	Found	Not Found	Not Found	Found
3	Gateway Mgmt interacting with GWFacade	1. Missing authentication	Not Found	Not Found	Not Found	Found
4	GWFacade interacting with DBMonitor	1. Missing authentication	Not Found	Found	Not Found	Found
5	NotificationHandler with DBMonitor	1. Missing authentication	Not Found	Found	Not Found	Found
6	NotificationController with CustomerDB	1. Missing authentication	Not Found	Not Found	Not Found	Found
7	Customer Mgmt with DBMonitor	1. Missing authentication	Not Found	Found	Not Found	Found
8	Operator Mgmt with DBMonitor	1. Missing authentication	Not Found	Not Found	Not Found	Found
9	Operator Mgmt with CustomerDB	1. Missing authentication	Not Found	Not Found	Not Found	Found
		2. Authentication Bypass using alternate path	1 False Positive			
10	Customer client to Customer Façade	2. Authentication Bypass using alternate path	Not Found	Not Found	Not Found	Not Found
11	Operator Client to Operator Façade	2. Authentication Bypass using alternate path	Not Found	Not Found	Not Found	Not Found
12	Gateway to Cloud	2. Authentication Bypass using alternate path	Not Found	Not Found	Found	Found
13	Mote to MicroPnP	2. Authentication Bypass using alternate path	Found	Found	Found	Found
14	Password for customer/operator	3. Relying on single factor Authentication	Found	Found	Found	Found
15	Customer client to Customer Façade	4. Insufficient Session Management	Found	Found	Found	Found
16	Operator Client to Operator Façade	4. Insufficient Session Management	Found	Found	Found	Not Found
17	Keys management not specified.	6. Insufficient Cryptographic Key Management	Found	Found	Found	Not Found
18	Customer - no authorization	7. Missing Authorization	Found	Found	Not Found	Found
19	Operator - no authorization	7. Missing Authorization	Found	Found	Not Found	Found
20	Gateway - no authorization	7. Missing Authorization	Not Found	Not Found	Not Found	Not Found
21	Mote - no authorization	7. Missing Authorization	Not Found	Not Found	Not Found	Not Found
22	Access control mechanism not specified	8. Missing Access Control	Found	Found	Found	Not Found
		9. Not reauthenticating before critical operation			1 False Positive	
23	UC4 Associate GW to Customer - no reauthentication	9. Not reauthenticating before critical operation	Found	Found	Not Found	Not Found
24	UC9 Rules Reconfiguration - no reauthentication	9. Not reauthenticating before critical operation	Found	Found	Not Found	Not Found
25	UC14 Changing SLA- no reauthentication	9. Not reauthenticating before critical operation	Found	Found	Not Found	Found
26	UC15 Unregister Customer- no reauthentication	9. Not reauthenticating before critical operation	Found	Found	Not Found	Found
		10. Unmonitored Execution of external Applications (Third party)		1 False Positive	1 False Positive	
		11. Not considering context when authorizing	1 False Positive	1 False Positive	1	1 False Positive
		12. Not revoking authorization				1 False Positive
27	Customer when unregistering - authorization not revoked	12. Not revoking authorization	Found	Found	Not Found	Not Found
28	Gateway when customer unregisters - authorization not revoked	12. Not revoking authorization	Not Found	Not Found	Not Found	Not Found
29	Operator when leaving company- authorization not revoked	12. Not revoking authorization	Not Found	Found	Not Found	Not Found
30	Customer Mgmt for configuring rules, SLA change, Deactivating account	12. Not revoking authorization	Not Found	Not Found	Not Found	Not Found
		13. Insecure Data Storage			1 False Positive	
31	SensorData in MainDB	13. Insecure Data Storage	Found	Found	Not Found	Not Found
32	SensorData in GWDB	13. Insecure Data Storage	Not Found	Not Found	Not Found	Not Found
33	Profile info in Customer/Operator DB	13. Insecure Data Storage	Found	Found	Not Found	Not Found
34	Logs in LogDB	13. Insecure Data Storage	Not Found	Not Found	Not Found	Not Found
		14. Insufficient Credentials management			1 False Positive	
35	Customer/Operator credentials storage, encryption, change mgmt	14. Insufficient Credentials management	Found	Found	Not Found	Found
36	Sensor Data from Mote to GW	15. Insecure data exposure	Not Found	Not Found	Not Found	Not Found
37	Sensor Data/Alarms from GW to GWFacade	15. Insecure data exposure	Not Found	Not Found	Not Found	Not Found
38	Sensor Data/Alarms from GWFacade to MainDB	15. Insecure data exposure	Not Found	Not Found	Not Found	Not Found
39	Sensor Data, Customer Info, View Logs from Customer/Operator Facade to Clients	15. Insecure data exposure	Not Found	Not Found	Not Found	Not Found
40	No encryption algorithms specified	16. Use of Custom/Weak Encryption Algorithms	Found	Found	Found	Found
41	Mote data used for alarms/events and stored in Main DB not validated	17. Not validating input/data	Found	Found	Not Found	Not Found
42	Gateway facade does not validate data from GWs	17. Not validating input/data	Found	Not Found	Not Found	Found
		18. Insufficient auditing			1 False Positive	1 False Positive
43	Logs have no back-up	18. Insufficient auditing	Found	Found	Not Found	Not Found
44	Motes to Gateway channel - no monitor, throttling or notification	19. Uncontrolled Resource Consumption	Not Found	Found	Not Found	Not Found
45	Gateway to Cloud channel - no monitor, throttling or notification	19. Uncontrolled Resource Consumption	Found	Found	Not Found	Not Found
46	NotificationController Channels - no monitor, throttling or notification	19. Uncontrolled Resource Consumption	Not Found	Not Found	Not Found	Not Found
47	CustomerClient to Customer Facade channel - no monitor, throttling or notification	19. Uncontrolled Resource Consumption	Found	Found	Not Found	Not Found
48	OperatorClient to Operator Facade channel - no monitor, throttling or notification	19. Uncontrolled Resource Consumption	Found	Found	Not Found	Not Found
49	GW DB capacity - no monitor or notification, possible to deplete	19. Uncontrolled Resource Consumption	Not Found	Not Found	Not Found	Not Found

Table B.1: Baseline of flaws for HomeSys.

C

STRIDE threat comparison with design flaws baseline

The analysis for each STRIDE threat is provided below. For each threat, the name, description, category, prerequisites, related flaws in the baseline and notes are included.

1. Threat Name: **Corrupt Mote**
   Description: Plug in a Mote that sends non-disruptive or no data to the gateway
   Category: S
   Prerequisites: Physical access to house to install or compromise Mote
   Related Flaws: none
   Notes: We cannot prevent threats where attacker has physical access or if a device is compromised.

2. Threat Name: **Spoofing mote**
   Description: Attacker sniffs IP and sends data to GW.
   Category: S
   Prerequisites: Sniff mote IP (channel), Access to internal network
   Related Flaws:
   - 1. Missing authentication -> Mote to MicroPnP component
   - 13. Authentication Bypass using alternate path -> Mote to MicroPnP

   Notes: Even if internal network is secured, Mote IP can still be obtained using a network auditing tool. Since the Mote does not authenticate it is easy to pretend to be a mote. STRIDE Spoof external entity tree (No authentication leaf)

3. Threat Name: **Overflowing the gateway with requests**
   Description: The attacker could overload the gateway with infinite number of requests to connect a new mote and cause a DoS for other already functioning motes.
   Category: DoS
   Prerequisites: Access to internal network
   Related Flaws:
   - 36. Insecure data exposure -> Sensor Data from Mote to GW
   - 44. Uncontrolled Resource Consumption-> Channel bandwidth Mote to GW

Notes: Traffic is not encrypted in the network between motes and GW. Attacker can easily capture a package and replay it. Domain description states that current limit for this network is 26 packets/second. There is no mechanism to monitor this or any throttling mechanism. No direct notification either. The gateway will fail to send updated data to the cloud so this will eventually trigger a notification to the customer and call center operators.

4. Threat Name: **Spoofing the gateway**
   Description: If the attacker sniffs the gateway IP and is able to send data to the cloud. This threat leads to others!
   Category: S
   Prerequisites: Sniff GW IP, access to internal network
   Related Flaws:
   - 2. Missing authentication -> GW to GWFacade(Cloud)
   - 12. Authentication Bypass using Alternate Path -> GW to Cloud

   Notes: Communication channel between GWs and GWFacade is not encrypted (not specified in documentation). Even if it is though, IP can still be obtained. Since GWs do not go through an authentication point when connecting with HomeSys cloud this attack is possible. This threat is captured by flaws 2,12. STRIDE Spoof external entity threat tree (No authentication leaf).

5. Threat Name: **Disable the watchdog**
   Description: The watchdog is initialized upon startup and keeps information about which sensors are connected. By removing the sensors from its list, the gateway watchdog would not expect heartbeats from it, and therefore it would fail to trigger an alarm if the sensor is not responding.
   Category: T
   Prerequisites: GW spoof or Elevation of Privilege
   Related Flaws: none
   Notes: List of motes in watchdog is not considered high value asset to us. In order to achieve this threat, it requires to either spoofing the gateway, compromise it or launch an EoP attack in its internal processes. These are not captured by the catalog, except for the spoofing of GW, but it is not exactly defined here.

6. Threat Name: **Disable the periodic scheduler**
   Description: If the component is corrupted, the cloud would not be synchronized accordingly. However, this would be detected directly by the customer and could be reported quickly.
   Category: T/DoS
   Prerequisites: GW spoof or Elevation of Privilege
   Related Flaws: none
   Notes: similar as previous threat.

7. Threat Name: **Tampering with GW DB**
   Description: If the attacker is able to write to Gateway DB and modify the

data store from the sensors, the rules will evaluate over false data and the cloud would receive false information, making it possible to hide alarms.
Category: T
Prerequisites: GW spoof or Elevation of Privilege
Related Flaws: none
Notes: This depends on how exactly the GW is spoofed, meaning which internal process is spoofed in the gateway. As for elevation of privilege, it is not captured by the catalog in this case, since attacker would have to elevate the privileges of a certain process in the GW, which means that the device would be compromised. Such an EoP attack would probably be due to implementation level bugs (buffer overflow, input validation etc). Could be captured by 32. Insecure Data Storage -> SensorData in GWDB

8. Threat Name: **False rules**
   Description: The rule engine managing the rules could be tampered with.
   Category: S/T
   Prerequisites: GW spoof or Elevation of Privilege
   Related Flaws: none
   Notes: similar to previous threat.

9. Threat Name: **Spoofing the DB Monitor**
   Description: The attacker would be able to change configurations of customers and effect the integrity of the copied secondary storage.
   Category: S
   Prerequisites: Elevation of Privilege (gain admin rights)
   Related Flaws: none
   Notes: Although Catalog captures missing authentication of processing regarding access to mainDB, since the prerequisites determine that attacker has system admin rights, there is nothing that we can do. According to documentation, operators pretty much have authorization to do anything in the system.

10. Threat Name: **DoS on Primary DB**
    Description: If the attacker has spoofed the DB Monitor first, they would be able to overwhelm the primary DB with requests and cause a DoS.
    Category: DoS
    Prerequisites: Spoof DBMonitor
    Related Flaws: none
    Notes: Notes: According to the documentation, the system is hosted in cloud and the DBMonitor along with the primary DB and the hot spare are place within the backend (assume same trust boundary). Since it is hosted on the cloud we assume storage capacity and network bandwidth are hard to deplete. Regardless, although entry 19. Uncontrolled Resource Consumption could capture the flaw, if DBMonitor, which is the mechanism supposed to monitor the DBs, is compromised, then there is nothing to do.

11. Threat Name: **Accessing sensitive info from DBs**
    Description: The attacker may attempt to expose private information of sensor data in primary DB, secondary DB, customer DB or operator DB about customers.
    Category: I
    Prerequisites: Elevation of Privilege
    Related Flaws: none
    Notes: 7. Missing Authorization and 8. Missing Access Control could capture this threat. However again if attacker elevates privileges to operator he can do anything. Similar to previous threat.

12. Threat Name: **Tampering with DS (Sensor data storage DB)**
    Description: Tamper with the primary SensorData DB and alter data.
    Category: T
    Prerequisites: EoP
    Related Flaws: none
    Notes: Notes: same as above.

13. Threat Name: **Spoofing the customer**
    Description: If the attacker pretends to be the customer, it can modify all configurations.
    Category: S
    Prerequisites: Steal secrets, MITM, Session hijacking
    Related Flaws:
    - 10. Authentication Bypass using alternate path-> CustomerClient to CustomerFacade
    - 14. Relying on single factor Authentication-> Password easy to brute force
    - 15. Insufficient Session Management-> Customer client to Customer Façade
    - 35. Insufficient Credentials management -> Customer/Operator credentials Storage,encryption, change etc
    - 39. Insecure data exposure-> Sensor Data, Customer Info, View Logs between Customer/OperatorFacade and Clients

    Notes: Overall, although session ID is secure and the auth token hard to guess based on the documentation, there is no limit on failed attempts to login, no session timeout, no session invalidation on logout mentioned etc. Password can be brute-forced since there is no limit in failed attempts. Channel encryption not specified, which means system is open to a number of MITM and session hijacking attacks.

14. Threat Name: **Spoofing the operator**
    Description: If the attacker pretends to be the operator, it can modify all configurations of clients.
    Category: S
    Prerequisites: Steal secrets, MITM, Session hijacking

Related Flaws:

- 11. Authentication Bypass using alternate path-> OperatorClient to OperatorFacade
- 14. Relying on single factor Authentication-> Password easy to brute force
- 16. Insufficient Session Management-> Operator client to Operator-Façade
- 35. Insufficient Credentials management -> Customer/Operator credentials Storage,encryption, change etc
- 39. Insecure data exposure-> Sensor Data, Customer Info, View Logs between Customer/Operator Facade andClients

Notes: Same as previous threat

15. Threat Name: **Spying on customers**
Description: The attacker can either be a valid operator and spy on customers, or an external attacker that has overtaken an identity of the operator. Even though actions of operators are recorded, this is a threat to privacy of customers.
Category: I
Prerequisites: Obtain credentials of a user/operator
Related Flaws:
    - 11. Authentication Bypass using alternate path-> OperatorClient to OperatorFacade
    - 14. Relying on single factor Authentication-> Password easy to brute force
    - 16. Insufficient Session Management-> Operator client to Operator-Façade
    - 35. Insufficient Credentials management -> Customer/Operator credentials Storage,encryption, change etc
    - 39. Insecure data exposure-> Sensor Data, Customer Info, View Logs between Customer/Operator Facade andClients

    Notes: Notes: similar as above (spoofing operator)

16. Threat Name: **Customer elevating privileges**
Description: This threat relates to either: elevating privileges from regular customers to premium customers, any type of customer to operator privileges, or higher - system administrator
Category: EoP/I
Prerequisites: The attacker needs to gain access to the DBs (customer or operator) and be able to modify type of account.
Related Flaws:
    - 18. Missing Authorization -> Customer
    - 22. Missing Access Control-> no mechanism specified
    - 25. Not reauthenticating before critical operation-> Changing SLA
    - 33. Insecure Data Storage-> profile info in Customer/Operator DB

Notes: Documentation doesn't specify the policy to assign privileges and although an access control mechanism is mentioned at some point there is no further clarification in the architecture. So we assume there isn't any authorization or access control mechanism specified. Changing SLA is considered a critical operation since it can have an impact on the customer profile (high value asset) and the sensor data(history and how long they are stored changes). Additionally, the profile info in the customer and operator DBs is not securely stored.

17. Threat Name: **Generating login requests**
Description: The attacker can cause a denial of service for a customer or operator by generating infinite login requests
Category: DoS
Prerequisites: -
Related Flaws:
    - 15. Insufficient Session Management-> Customer client to Customer Façade
    - 16. Insufficient Session Management-> Operator Client to Operator Façade
    - 39. Insecure data exposure-> Login, Sensor Data, Customer Info, View Logs between Customer/Operator Facade and Clients
    - 47. Uncontrolled Resource Consumption-> CustomerClient to Cloud (Customer Facade)
    - 48. Uncontrolled Resource Consumption-> OperatorClient to Cloud (Operator Facade)

    Notes: There is no mechanism to monitor session creation/destruction or any limit to how many endpoints can be connected to a session. Packets can be replayed in the channel between the clients and facades, and there is no mechanism specified to monitor the incoming connections or any throttling for that matter.

18. Threat Name: **Sniffing the credentials**
Description: The attacker can try to sniff the traffic and steal the credentials (preparation for spoofing)
Category: I
Prerequisites: -
Related Flaws:
    - 10. Authentication Bypass using alternate path-> Customer client to Customer Façade
    - 11. Authentication Bypass using alternate path-> Operator Client to Operator Façade
    - 15. Insufficient Session Management-> Customer client to Customer Façade
    - 16. Insufficient Session Management-> Operator Client to Operator Façade
    - 35. Insufficient Credentials management-> Customer/Operator creden-

tials Storage,encryption, change etc
- 39. Insecure data exposure-> Login, Sensor Data, Customer Info, View Logs between Customer/Operator Facade and Clients

Notes: Communication channels are not secure, credentials are not encrypted in rest or in transit. If session ID is used as a URL parameter not specified.

19. Threat Name: **SQL injection on client side**
Description: -
Category: T
Prerequisites:
Related Flaws:
Notes: SQL injection is technology specific. It is out of scope for the catalog.

20. Threat Name: **Disable logging operator actions**
Description: The operator can attempt to harm the accountability controller and disable logging of actions.
Category: T
Prerequisites: Operator is authorized
Related Flaws: none
Notes: Since operator according to documentation has full access, we cannot prevent that. The only case would be to have back-ups of the logs that would not be accessible to operators. (43. Insufficient auditing-> Logs have no Back-up)

21. Threat Name: **Generating view sensor data requests**
Description: The attacker can try to overwhelm the cloud by generating a lot of view data request, where a heavy load of sensor data has to be displayed.
Category: DoS
Prerequisites: he operator is authorized.
Related Flaws: none
Notes: According to our assumptions, since the system is hosted in the cloud, processing capacity shouldn't be an issue. Even then DBMonitor is supposed to monitor the DBs and handle this operation.

22. Threat Name: **Tampering with customer data**
Description: The operator can attempt to modify customer configuration without their request.
Category: T
Prerequisites: Operator is authorized
Related Flaws: none
Notes: Missing authorization and Missing access control could capture this, but again operator has full access according to documentation.

23. Threat Name: **Fill up the Event/Alarm queue**
Description: The attacker can attempt to send infinite event/alarms to the queue to overwhelm it so that it can't receive any more valid events/alarms.

Category: DoS
Prerequisites: The attacker needs to know how to publish to the queue. Alternatively, the attacker is an insider and creates rules that always evaluate, but this depends on the system implementation and queue response time.
Related Flaws: none
Notes: This threat depends heavily on implementation. Regardless, alarms are triggered in the gateways and are sent from the cloud. So this could be captured in a way from 45. Uncontrolled Resource Consumption-> Gateway to Cloud (Gateway Facade) but this is only based on assumptions.

24. Threat Name: **Fill up the sensor queue**
Description: Attempt to overwhelm the sensor data queue
Category: DoS
Prerequisites: The attacker needs to know how to publish to the queue.
Related Flaws: none
Notes: Same as previous threat

25. Threat Name: **Sniff the sensor data (GW to DBMonitor)**
Description: The attacker can attempt to sniff the traffic and retrieve private sensor data packages over the internet.
Category: I
Prerequisites: The attacker needs to redirect data in the network (e.g. DNS poisoning)
Related Flaws:
    - 37. Insecure data exposure-> Sensor Data/Alarms from GW to GWFacade
    - 38. Insecure data exposure-> Sensor Data/Alarms from GWFacade to MainDB

Notes: These channels are not secure and packets easily sniffed.

26. Threat Name: **Sniff events**
Description: Same as above
Category: I
Prerequisites: The attacker needs to redirect events in the network (e.g. DNS poisoning)
Related Flaws:
    - 37. Insecure data exposure-> Sensor Data/Alarms from GW to GWFacade
    - 38. Insecure data exposure-> Sensor Data/Alarms from GWFacade to MainDB

Notes: Same as previous threat

27. Threat Name: **Generate gateway connections**
Description: Attempt to generate fake connections to overrun the available capacity of connected gateways to the cloud.
Category: DoS

Prerequisites: The attacker would have to spoof the gateway or connect a corrupt device.
Related Flaws:

- 45. Uncontrolled Resource Consumption-> Gateway to Cloud (Gateway Facade)

Notes: There is no mechanism to monitor connections to GWFacade or any throttling mechanism.

28. Threat Name: **Attacking logs**
Description: The attacker spoofs the process of logging all actions, or is able to tamper with the log files in order to hide malicious activity.
Category: S
Prerequisites: Compromise system to get access rights to logs
Related Flaws:
    - 43. Insufficient auditing-> Logs have no back-up

    Notes: If process is spoofed then this can possibly be an issue, but if logs are tampered with, although DB storing logs is secure, there is no back-up.

29. Threat Name: **Tamper with sensor data before it reaches cloud**
Description: If the sensor data is sent in plain, the attacker can try to modify it not to raise any alarms or events.
Category: T
Prerequisites: The attacker needs to perform a MIM attack.
Related Flaws:
    - 37. Insecure data exposure-> Sensor Data/Alarms from GW to GWFacade

    Notes: DFD shows GW to Cloud so we assume GW façade. Channel is not encrypted, MITM is possible.

30. Threat Name: **Tamper with configuration in transit**
Description: The attacker may attempt to intercept the traffic and tamper with the rules before they are store in the data base of the customer.
Category: T
Prerequisites: The attacker needs to perform a MIM attack.
Related Flaws:
    - 37. Insecure data exposure-> Sensor Data/Alarms from GW to GWFacade

    Notes: same as previous threat

31. Threat Name: **Spoofing the cloud**
Description: Attempt to imitate take over the identity of the cloud and send false configuration data (rules), or not react upon alarms and events
Category: S
Prerequisites: The attacker would have to authenticate to the gateway as the cloud service and hide all its activity to the actual cloud (not to raise suspicion)
Related Flaws:

- 2. Missing authentication->Gateway connecting to Cloud
- 3. Missing authentication->Gateway Mgmt interacting with GWFacade
- 12. Authentication Bypass using alternate path->Gateway to Cloud

Notes: No authentication between GWs and cloud, or the process handling the rule configuration (Gateway Mgmt) with GWFacade

32. Threat Name: **Reveil sensor data or tamper with rules**
Description: The attacker may attempt to intercept the sensitive sensor data being stored in the local DB or read from the local DB.
Category: I and T
Prerequisites: The attacker had to gain control over GW- spoofed it before.
Related Flaws: none
Notes: Similar to Threat number 7

33. Threat Name: **Block/redirect data to cloud**
Description: The alarm data is send from the gateway to the gateway facade via the internet, which is generally insecure. Attacker may attempt to redirect traffic to block the alarm data.
Category: DoS
Prerequisites: The attacker needs to perform a DNS poisoning, or spoofing attack.
Related Flaws:
    - 37. Insecure data exposure-> Sensor Data/Alarms from GW to GWFacade

Notes: Channel from GW to Cloud is not secure and leaves system vulnerable to MITM attacks, sniffing etc.

34. Threat Name: **Blocking notification channels**
Description: Attempt to disrupt the communication channels via SMS, email or App push. Could also do so, by filling up the message queues capacity.
Category: DoS
Prerequisites: Fill message queue, compromise phone, sms protocol or redirect emails
Related Flaws:
    - 46. Uncontrolled Resource Consumption-> NotificationController Channels

Notes: No monitor mechanism, or throttling

D

Questionnaire

Below the questionnaire given to the participants of our study to fill in after the task is presented.

Design Flaws as Security Threats Survey

*Required

1. **How easy was the process to understand?** *
   Mark only one oval.

	1	2	3	4	5	
Difficult to understand	◯	◯	◯	◯	◯	Easy to understand

2. **Did the guidelines provide enough information to detect the flaw?** *
   Mark only one oval.

	1	2	3	4	5	
Little information	◯	◯	◯	◯	◯	Sufficient information

3. **Was any of the guidelines ambiguous?** *
   Mark only one oval.

   ◯ Yes
   ◯ No

4. **What type of models did you use throughout the process?** *

5. **Would you require another type of UML model?** *

6. **How easy was it to get all the information required by the architecture to follow the guidelines?** *
   Mark only one oval.

	1	2	3	4	5	
Very hard	◯	◯	◯	◯	◯	Very easy

7. **Do you think the process required a lot of time?** *

8. **Do you have any suggestions to reduce the time required?** *

9. **How familiar were you with the architecture before using the catalog?** *
   Mark only one oval.

	1	2	3	4	5	
Very unfamiliar	◯	◯	◯	◯	◯	Very familiar

10. **How familiar are you with the domain and its constraints, regulations etc?** *
    Mark only one oval.

	1	2	3	4	5	
Very unfamiliar	◯	◯	◯	◯	◯	Very familiar

11. **Any suggestions for improving the detection guidelines?**

12. **Have you performed threat analysis with STRIDE before?**
Mark only one oval.

- Yes
- No

13. **If yes, compared to STRIDE, how would you rate this approach in terms of time and efficiency and why?**

14. **What aspects of the approach did you like?**

15. **What aspects of the approach did you not like?**

E

Documents provided in the lab package

The documents presented in this Appendix are introduced in the course DAT220 Advanced Software Architecture in the Software Engineering MSc programme in Chalmers University of Technology.

E.1 HomeSys Domain Problem Description

Home Monitoring System (HomeSys) – Description of the problem domain

DAT220/DIT544 - Advanced Software Architecture

2016

This project requires the development of a smart home system code-named HomeSys. The system consists of a smart home gateway that can communicate with plug and play sensors spread around the house, and a cloud system that synchronizes with the smart home gateway.

This document describes the project, the problem space and the additional constraints. Sections 1 and 2 provide some background about the domain of smart home systems and a description of how as we envision HomeSys. Section 3 lists the key stakeholders involved in the development of this system, and Section 4 describes a number of additional constraints the system should comply to. Finally, Section 5 elaborates on some more specific scenarios.

1 Context

Home automation

Home automation is the residential extension of building automation. It is automation of the home, housework or household activity to provide improved convenience, comfort, energy efficiency and security. Home automation may include centralized control of lighting, HVAC (heating, ventilation and air conditioning), appliances, locks of gates and doors and other systems. The popularity of home automation has been increasing greatly in recent years due to much higher affordability and simplicity through smartphone and tablet connectivity. The concept of the"Internet of Things" (IoT) has tied in closely with the popularization of home automation.

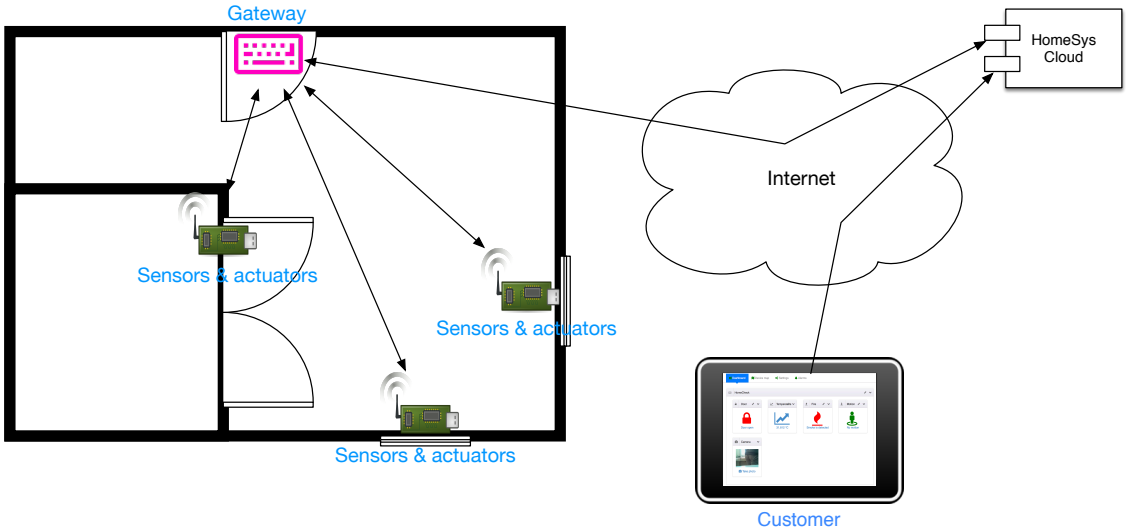


Figure 1: HomeSys overview

Figure 1 provides a schematic overview of a typical home automation system. **Sensors** and **actuators** are installed throughout the house and relay data to a cloud-based system via a **Gateway**. The cloud-based services are accessible to the customer via a tablet app or a browser and

provide the user with a management console as well as an information dashbord displaying the historical data produced by the sensors in the house. The customer can both monitor his house as well as control the actuators available (e.g., dim the lights, shut off the water supply). For certain alarming events (e.g., in case smoke is detected) the user can get immediate notifications.

Examples from the competitors

A number of commercial implementations for home automation systems are readily available on the market.

NEST

Perhaps the most widely known implementation of the home automation systems is NEST which has been acquired by Alphabet Inc. (the mother company of Google). NEST features a smart thermostat, a smoke/CO2 sensor and a camera. All NEST devices connect directly to the home WiFi router which serves as a gateway.

Philips Hue

Philips Hue is mainly focused on smart illumination. The sensors/actuators are in fact Philips lightbulbs with integrated actuators. The Philips Hue gateway is called a "bridge".

Belkin WeMo

Belkin WeMo mainly consists of lights and switches which are pluggable in various devices (e.g., the coffee maker). The Belkin gateway is in fact a wireless router Belkin produces.

Samsung Smart Things

Samsung Smart Things is probably the most extended implementation of home automation available today. The gateway here is referred to as the Smart Things Hub and it is interoperable with a large number of existing smart objects like light bulbs, cameras, speakers, locks, and thermostats.

Positioning

In this assignment, we assume the point-of-view of a software company that will develop a home automation gateway and its cloud-based services. The system is initially being built for the remote monitoring of residential apartments and houses. However, it would be beneficial if the same system could be easily extended for other purposes, e.g., monitoring of server centers, storage facilities, etc. Our target customers are people who would like to keep an eye on their apartment or house.

Our goal is to sell the HomeSys gateway as a hardware device enabling smart home automation. A basic cloud-based subscription will be offered with the device, free of charge. However, the basic subscription will only allow the customers to have data history (from sensors' data) up to 1 month and the gateway-cloud communication might be delayed (i.e., the sync will run every 15 minutes). We will also offer a premium plan (for a yearly fee) which will include data history up to 1 year and virtually a real-time communication between the gateway and the cloud services.

The main differentiation point with the existing technology providers will be twofold. First, the HomeSys gateway will feature a 3G communication module, hence the module will be 100% plug and play and it will not require any pre-existing infrastructure. Second, the gateway will be designed with explicit focus on incorporating new types of sensors and actuators, in order to be future-proof.

Technology stack

From a technical perspective, the HomeSys system leverages the following building blocks.

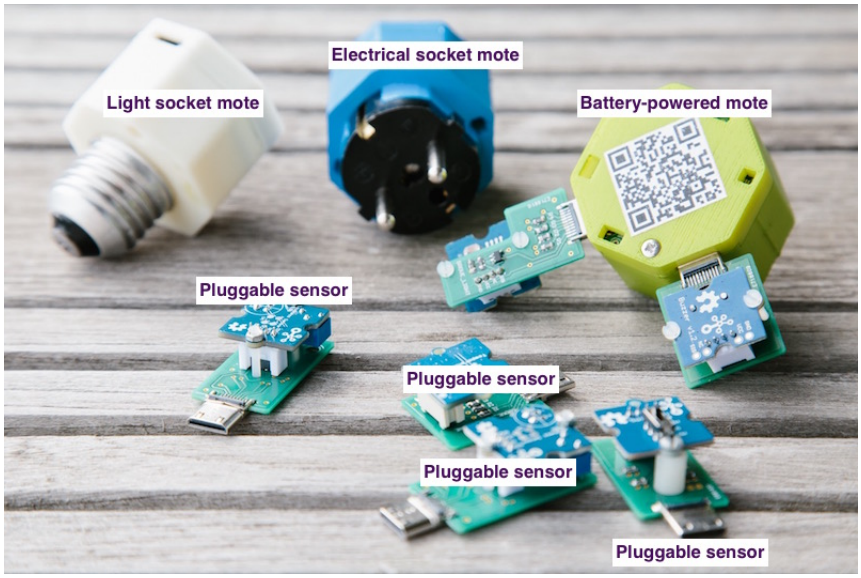


Figure 2: MicroPnP motes and sensors

MicroPnP mote (or just mote) is a hub where the sensors and actuators plugged onto and installed throughout the house. The MicroPnP technology is developed by iMinds-DistriNet (a research lab in Belgium)[1]. As presented in figure 2 motes are powered either by a battery or they can be plug into an electrical or light socket. Each MicroPnP mote has 4 USB-like ports where sensors and actuators can be plugged into. In figure 1 the sensors are represented as the green printed circuit boards (PCBs) with antennas.

Sensor is a hardware device that produces measurements and sends them to the gateway via a MicroPnP mote. The sensors are physically plugged into a mote (see figure 2).

Actuator is a hardware device that has one or more actions associated with it. For instance, a camera features "take picture" and "take video" actions, while a switch has a "turn on" and "turn off" actions. These actions are triggered by the gateway. The actuators are also plugged into the MicroPnP motes.

Figure 3: HomeSys gateway

HomeSys gateway is a pluggable black-box device that requires no technical knowledge for the installation. Figure 3 shows how the HomeSys gateway might look like. The gateway relays

[1]For more information on MicroPnP motes we refer to a very short video available on http://www.micropnp.com/ipso/index.html

the sensor data to the cloud and manages the actuators. The gateway is a Linux platform running software that our firm will develop. The gateway hardware is sufficiently powerful to run a high-level execution environment (e.g., to run Java/Python programs) and a modern database management server (e.g., MySQL). The gateway is typically installed at the front door of an apartment or a house (the pink icon on figure 1). It connects to the HomeSys cloud services via a 3G dongle (although connectivity via the home WiFi router is also feasible).

HomeSys cloud is a software system including a portal where the customers of our firm can log in and view the information about their houses.

Sensors installed on motes are also immediately available to the gateway, which can communicate with them using JSON messages. The MicroPnP supports four types of events.

1. **Heartbeat.** The motes send regular heartbeats to the gateway indicating that they are alive.
2. **Sensor plugged in.** Whenever a new sensor is inserted in the mote, a message is sent to notify the gateway that a new sensor was added.
3. **Data.** The motes send information from the sensors.
4. **Sensor removed.** Whenever a sensor is removed, the mote sends this information to the gateway.

Only the HomeSys Gateway and the HomeSys Cloud services need to be designed (as part of the assignment) as the remaining building blocks are provided by third parties.

2 Description of the problem domain

2.1 Overall system goals

The main goal of the system is to provide the customers with the necessary tools to monitor their home and to get prompt notifications upon critical events (e.g., smoke, low / high temperature, etc.). Initially our firm will work with a fixed set of sensors, i.e., temperature, humidity, smoke/gas, motion, open/closed doors and windows. However, in the future new types of sensors and actuators can be plugged in so that the customers can have much more thorough information about their home. For instance, water leak sensors, valves to turn on/off the water supply, actuators to turn on/off electrical devices, and so on.

In order to achieve the primary goals, HomeSys should realize (at least) the following objectives.

- **Home monitoring.** All sensor data should be synchronized (i.e. relayed) to the HomeSys Cloud where the customers can view all the information relevant to their home (possibly, in a summarized way).
- **Home management.** Customers should be able to manage their actuators as well as configure actuator actions depending on certain sensor value readings.
- **Usability and user friendliness.** HomeSys should be completely plug and play requiring no configuration[2].

The remainder of this section zooms in on these objectives.

[2]In case of the WiFi connectivity, the configuration to make the HomeSys gateway connect to the home WiFi network will be required though.

2.1.1 Home monitoring

The HomeSys gateway should store the sensor data internally and synchronize with the HomeSys Cloud system regularly. It is essential that the synchronization protocol works correctly in the presence of non-reliable communication network. The 3G network could switch to Edge and that would mean that the synchronization protocol could easily get timeouts or network errors. While basic sensor data (e.g., temperature, humidity readings) can be packaged in very small units of transfer, image data coming from the camera could be fairly large. Aside from simply syncing the sensor data, the HomeSys gateway should also monitor the health status of each sensor. Sensors could break or they could start sending faulty data. MicroPnP motes that are working on a battery could run out of battery and stop sending data. All this status information should be monitored by the HomeSys gateway and in case of problems the customer should get an alarm. For a more detailed description regarding alarms we refer to the next subsections. All the monitoring data will be available for the customers to consult at any time via the cloud-based services. For instance, the customer should be able to look at the last pictures taken by the cameras, or view the graph of the temperatures in the house rooms during the previous days (i.e., historical data).

2.1.2 Home management

We would like to allow the customers to easily create specific **events** of interest they would like to be notified about. These events are in fact based on specific sensor data values. Some examples of events are a temperature value that goes above 35 degrees, motion detected at night, an opened door, etc. So the customer should be able to receive **notifications** when these events are detected by the gateway and they should be able to define **rules** in the HomeSys cloud about how to handle the events (e.g., taking a picture when motion is detected).

Aside from simply defining events the customer should be able to indicate that some events are alarming (e.g., smoke sensor indicating smoke, very high or very low temperatures, very high or very low humidity values, etc.). **Alarms** should be handled faster than regular sensor data and notifications. Customers should be notified immediately upon alarms.

As mentioned, the MicroPnP motes also can have actuators (e.g., buzzer, camera, on/off switch). Each actuator type has a set of actions/states that it can take. These actions/states are pre-defined. For instance, a buzzer has “on” and “off” actions, a camera has a “take picture” action. Customers should have the possibility to define rules that link events to actuator actions/states. For instance the event “motion is detected” could be linked to the camera “take picture” action, or “door open” could be linked to a light switch “on” action.

All these rules are managed by the customer on the HomeSys cloud and synchronized to the gateway regularly.

All notifications are handled by the HomeSys cloud. This means that notifications and alarms are sent by the gateway to the cloud. The cloud then will notify the customer.

Actuators always have a certain lag before their state changes. While some actuators are pretty much instant (e.g., buzzer on / off), others may require some time before they complete the action (e.g., camera shutter speed). This means that an action will be acknowledged twice, i.e., once when received and once when successfully completed. The gateway will need to make sure that successive actions are queued and do not interfere with each other.

2.1.3 Usability and user friendliness

Existing systems similar to HomeSys require a relatively complex installation procedure. Something that most elderly people who are rich enough to have a second home are unwilling to accomplish. Thus the PnP goal has three dimensions:

- As soon as HomeSys gateway is plugged into the electrical socket it immediately connects to the Cloud and starts sending data.
- As soon as a new MicroPnP mote is powered up it is instantly recognized by the HomeSys.
- Sensors that are plugged in (or out) into the MicroPnP motes should also appear instantly within the HomeSys.

These goals are completely feasible from the technological point of view. For instance a 3G module allows HomeSys to require no configuration to get online. MicroPnP motes broadcast information as soon as they are powered. The broadcasted information contains both meta information regarding the sensors that are available, as well as sensor data. The only step the customer will have to do is a registration procedure on the HomeSys cloud and a linking of the gateway to the customer account.

For simplicity reasons we currently assume that each HomeSys gateway and its MicroPnP motes do not interfere with other HomeSys environments. This means that you don't have to worry about two neighbours each having a HomeSys gateway at home.

2.2 Sensors and Actuators

In this section, we briefly describe the sensors and actuators that we will use initially. Note that in general there are two types of sensors, i.e., analog and digital. The analog sensors require an explicit action from the MicroPnP mote (or the gateway) to get a value from the sensor. The digital sensors trigger an event whenever they sense a new value (or a state change). Such a triggered event is typically linked to what we refer to as callback. The callback is simply a low-level method that implements the business logic of processing that specific type of sensor data.

Temperature and humidity sensor. This is an analog combined sensor that samples the environment temperature and humidity. The temperature reading has a range from -30C to +85C. The humidity reading has a range from 0-100% relative humidity.

Smoke and CO2 sensor. This is a digital sensor that triggers an event in case of a smoke. Only when a smoke / CO2 is detected the sensor will spring alive. It will also report when the smoke/CO2 has stopped. It is possible to configure the sensitivity of this sensor. In the first version this is done using a hardware screw. However, in the future we expect to work with smoke sensors that will allow adjusting the sensitivity via the software (by sending actuator commands).

Motion sensor. This is a digital sensor that invokes a callback when motion is detected and when no motion has been detected after a motion. The motion detector operates on the Infrared frequency (so light is not strictly necessary to detect motion). It is possible to configure the sensitivity settings of the motion sensor.

Door/window sensor. This is a digital sensor that invokes a callback when the door/window opens or closes. The door/window sensor typically consists of 2 magnets and a wire running from one of the magnets all the way to the sensor plugged into the MicroPnP mote.

Camera actuator. This is a camera that can take pictures or video's on request. The request to take a picture, for instance, is not blocking. It means that the camera will callback when the picture or video is ready. The camera will give an error if two processes try to simultaneously take a picture or a video.

Buzzer actuator. This is an actuator that takes two commands, i.e., start and stop the buzzer.

Switch actuator. This is an actuator that can be hooked up to a variety of devices and turn them on and off using the appropriate commands.

2.3 Service Level Agreements

Our firm offers a Service Level Agreement (SLA) enclosed as a part of the provided support towards our customers. This agreement stipulates certain quality requirements (in terms of performance and availability). For example, there should be an upper limit in the time it takes for a notification to reach its recipient. Evidently the upper limit for alarm notifications should be much more strict than for normal notifications (e.g., 95% of alarm notifications should reach the recipient within 30 seconds, and 95% of event notifications should reach the recipient within 5 minutes).

Aside from the notification/alarm SLA, our firm will also provide different SLAs for customers. For instance, SLA level 1 will allow customers to access their data only for the past month. The

synchronization of data between the gateway and the cloud will take place every 15 minutes. SLA level 2 customers will be able to access the data for the past 6 months. The synchronization of data between the gateway and the cloud will take place every 5 minutes. SLA level 3 customers will be able to access their data for the past years. The synchronization of data between the gateway and the cloud will happen every minute. Customers can easily move from one SLA level to another. Hence, the SLA information should be synchronized to the gateway as well.

Also, our company has to negotiate a number of service level agreements with several third parties, like the telecom operators, to ensure the desired degree of remote monitoring module connectivity.

3 Main stakeholders

This section provides an overview of the main stakeholders in our system and their point of interest.

Customers. The customers' main stake in the system is that they would like to have a tool to monitor their home, define events of interest in a user-friendly fashion, get prompt notifications upon these events and link these events to specific actions. They would also like to add to plug new sensors/actuators to their HomeSys installation. Obviously, security and privacy related concerns are of the utmost importance to them.

MicroPnP mote manufacturer. The MicroPnP motes will be hubs for plugging multiple sensors and actuators. These motes will be linked to the HomeSys gateway via a mesh-up.

HomeSys call center. The HomeSys call center employs HomeSys operators that should follow up on various tasks. The operators take on every sort of question about the system from the customer's side and help them tackle any issues with the installation and configuration. The call center is a service offered by our company.

HomeSys system administrator. The HomeSys system administrator monitors the cloud system for its correct working. They are responsible for the scalability of the cloud system. They are notified in case of any alarming events.

Telecom operators. Telecom operators provide means for the gateways to communicate with the cloud and vice versa. A service level agreement between our company and the telecom operators determines the capabilities of these communication channels.

Cloud providers. The cloud providers provide virtual servers on the cloud that will be used to deploy the HomeSys cloud services. A service level agreement between our company and the cloud providers determines the server uptime constraints as well as the reaction time in case of problems.

Market researchers and statisticians. Given that the HomeSys produces a vast amount of data, this data could be used in various types of market research and the generation of certain statistics. For instance, it might be of interest to know the average temperature in houses around the year. This data will be sold by our firm in an anonimized format.

4 Additional constraints

4.1 Legal constraints

As the HomeSys gateway will feature a 3G connectivity it is essential to ensure all **regulations regarding the SIM card** that will be installed on the gateway. End customers will have to explicitly agree to terms and conditions regarding the use of the SIM card. This will make sure that the end users are liable for any misuse of the SIM cards. Our firm is also likely to consider

to use SIM cards from the cheapest mobile operators across EU (as the roaming costs are likely to be dropped after 2017). This might also have some legal implications regarding the end customer private information (such as, social security number and address).

Further, the **privacy** of the customer personal data as well as the sensor data need to be preserved in order to avoid leaks. This information needs to be transmitted confidentially, stored securely and preserved from unauthorized access. The customers also need to agree to the privacy policy of our firm.

4.2 MicroPnP constraints

The MicroPnP motes are constraint by the trade-off between latency, energy and bandwidth. Certain MicroPnP motes are battery operated which means that they sleep most of the time. The lower the latency of the motes, the higher the energy consumption. MicroPnP motes have a mesh network topology (i.e., data could be relayed through intermediary motes that are sometimes only used as intermediaries). This has an impact on the bandwidth that is currently limited to roughly 26 packets per second. The lower the latency of the motes the less the bandwidth.

5 Scenarios

In this section, we present a set of concrete scenarios concerning the typical installation and usage of the gateway and the cloud services. The presented scenarios involve a representative customer named John Appleseed who has a summer house in Marstrand, on the seaside of western Sweden.

5.1 HomeSys gateway installation

John buys in a store a HomeSys 3G gateway. John plugs the gateway into an electrical outlet somewhere in his house. Once the gateway is powered, it immediately connects to the cloud and sends information regarding its configuration. The cloud recognizes the HomeSys by its serial number and saves the connection timestamp and the initial configuration.

5.2 Customer profile creation

John goes to the HomeSys cloud and creates an account. The account creation requires John to enter the serial number of the HomeSys gateway. If the serial number is not found within the HomeSys cloud the account creation is disallowed. Otherwise John's account gets successfully created and from this moment on John can log in into his account.

5.3 MicroPnP mote installation

Together with the gateway John also buys a MicroPnP mote. He plugs the MicroPnP mote in his bedroom into the power socket. The mote immediately sends a broadcast notification which is received by the gateway. The gateway stores the information and upon the next next data transmission to the cloud includes the new mote information.

5.4 Sensor plugged in into MicroPnP mote

John buys temperature, humidity, smoke and motion sensors. He plugs them in one by one into the installed MicroPnP mote. The mote immediately recognizes the new hardware and sends this information to the gateway. The gateway stores the new sensor information and synchronizes this information to the cloud as well.

5.5 Actuator plugged in into MicroPnP mote

John buys a second MicroPnP mote which he installs above his house entrance door (see scenario 5.3). John buys a camera, a light switch and a buzzer. He plugs them in one by one into the newly installed MicroPnP mote. The mote immediately recognizes the new hardware and sends this

information to the gateway. The gateway stores the new actuator information and synchronizes this information to the cloud as well.

5.6 Sensor moved

John decides to move the smoke sensor from the bedroom to the entrance door mote. So he plugs the smoke sensor out of the MicroPnP mote in the bedroom. The mote immediately senses this and notifies the gateway. The gateway will store this information and notify the cloud that the smoke sensor is no longer used in the bedroom mote.
Then John plugs the smoke sensor in the mote above the entrance door (see scenario 5.4).

5.7 MicroPnP analog sensor data

Temperature and humidity sensors are analog sensors that periodically sample the environment for temperature and humidity values. MicroPnP periodically invokes the temperature / humidity sensors to sample the environment and sends this data to the gateway. The gateway stores this data.

5.8 MicroPnP digital sensor data

Smoke and motion sensors are digital which means that whenever the sensor value changes (e.g., from no smoke to smoke, or from no motion to motion detected) the sensor invokes a callback on the MicroPnP. The MicroPnP sends the new value to the gateway where this information is stored.

5.9 Normal sensor data transmission

At regular intervals the gateway (according to its configuration) sends data to the HomeSys cloud. By data we mean the following:

- information on which motes are installed,
- information about the sensors that are installed on each mote,
- information about the actuators that are installed on each mote,
- the sensor data,
- the actuator states.

Note that it is up to you to decide which data to send and how. For instance, you may send all this information on every transmission, or you could send only the new information. On every transmission the cloud can also send data back to the gateway (e.g., configuration data).

5.10 Data viewing

John logs in into his account on the HomeSys cloud and gets a dashboard with all his sensor data. For each sensor type he can request a more detailed view of the data.

5.11 Notification profile

John logs in into his account and configures an email address for all notifications and a mobile phone number for all alams (see scenario 5.12).

5.12 Event configuration

John logs into his account and creates three event configurations. The first event is a simple notification upon motion at night. The second event is a door opening that is then linked to the camera to take a picture. Finally, the third event is a smoke alarm that is linked to the buzzer on action. The last event is marked as an alarm event. This configuration data is sent to the HomeSys gateway, e.g., when the gateway connects to transmit sensor data (see scenario 5.9). The gateway receives this information and stores it.

5.13 Event notification transmission

A HomeSys gateway motion sensor records motion at 3AM and a notification is created and sent within a specific timeframe to the cloud. The cloud receives this notification and sends an email to the email address provided by John.

5.14 Alarm transmission

John's dad couldn't fall asleep and went for a smoke. Not knowing anything about the new gadget, he lighted up a cigarette before leaving the house, which triggered the smoke alarm. HomeSys gateway sends this info to the cloud. The cloud in turn saves this data and sends an SMS to John.

5.15 Actuator action

John has changed the configuration from the cloud so that his gateway will always take a picture when the door is opened. Upon such an event the gateway triggers the camera to take a picture. The camera picture is stored as sensor data and it is linked to the trigger event. When John checks his data on the cloud (see *Data viewing*) he can see the picture linked to the door opening.

5.16 Actuator malfunctioning

The gateway sends a new actuator action to one of the MicroPnP motes to switch off the light. The mote acknowledges that the command was correctly received, however after some time the mote still fails to report that the command was successfully executed. The gateway tries again, but after several failed attempts the actuator is flagged as malfunctioning and a new alarm is created and sent to John (via the cloud).

E.2 HomeSys Requirement Specification

Home Monitoring System (HomeSys) – Requirements

DAT220/DIT544 - Advanced Software Architecture

2016

Contents

1 Functional requirements **2**

1.1 *UC1*: Log in . . . 2
1.2 *UC2*: Customer profile creation . . . 2
1.3 *UC3*: Customer profile activation . . . 3
1.4 *UC4*: Associate gateway to customer . . . 3
1.5 *UC5*: Send configuration to gateway . . . 4
1.6 *UC6*: Add a mote to the customer home . . . 5
1.7 *UC7*: Plug in a sensore in a mote . . . 5
1.8 *UC8*: Send sensor data to the cloud . . . 6
1.9 *UC9*: Rule (re)configuration . . . 6
1.10 *UC10*: Evaluate rules on gateway . . . 7
1.11 *UC11*: Notify customer . . . 7
1.12 *UC12*: View sensor data in the cloud . . . 7
1.13 *UC13*: View events and alarms in the cloud . . . 8
1.14 *UC14*: Changing of service level agreement . . . 8
1.15 *UC15*: Unregister customer . . . 8
1.16 *UC16*: Customer invoicing . . . 9

2 Quality requirements **10**

2.1 *QAS1*: HomeSys Cloud sensor data DB failure . . . 10
2.2 *QAS2*: HomeSys Gateway not sending data to the Cloud provider . . . 10
2.3 *QAS3*: Timely notification of alarms . . . 11
2.4 *QAS4*: Large number of Customers . . . 11
2.5 *QAS5*: Authentication of Cloud users . . . 12
2.6 *QAS6*: Users are accountable for their actions . . . 12

1 Functional requirements

1.1 *UC1*: Log in

- **Primary actor:** Customer/Operator
- **Interested parties:**
  - *HomeSys Cloud:* wants to authenticate its Customers/Operators for access control and traceability.
  - *Customer/operator*: wants to be able to use HomeSys Cloud.
- **Preconditions:**
  - The Customer/Operator is registered into the system and has credentials to prove their identity.
- **Postconditions:** The Customer/Operator has authenticated themself in HomeSys Cloud.
- **Main scenario:**
  1. The Customer/Operator indicates they want to authenticate into HomeSys Cloud.
  2. HomeSys Cloud asks them to provide their credentials (e.g., email and password).
  3. The Customer/Operator provides their credentials.
  4. HomeSys Cloud verifies that the provided credentials are correct and authenticates the Customer/Operator.
- **Alternative scenarios:**
  1. 4b. The provided credentials are incorrect, resume at step 2.

1.2 *UC2*: Customer profile creation

- **Primary actor:** Customer
- **Interested parties:**
  - *HomeSys Cloud*: wants its customers to create profile in order to monitor their home.
  - *Customer*: wants to gain access to HomeSys Cloud.
- **Preconditions:**
  - Customer needs to have the serial number of the HomeSys Gateway.
- **Postconditions:**
  - The Customer is added to the HomeSys Cloud list of users and his profile is created.
  - A HomeSys Gateway is now associated to a Customer.
- **Main scenario:**
  1. The Customer indicates they wants to start the registration process.
  2. HomeSys Cloud asks the Customer for the profile information. This includes at least:
     (a) Primary email and password (credentials)
     (b) Full name and address
     (c) Cell phone number and e-mail address (for notifications)
     (d) Type of subscription (e.g., gold, silver, bronze)
     (e) Timezone of the customer
  3. The Customer enters the requested information.

4. HomeSys Cloud creates the profile for the new Customer
5. HomeSys Cloud sends a confirmation to the Customer (e-mail or SMS, **Include:**. *UC11: Notify customer*)
6. HomeSys Cloud marks the profile as 'pending' until a confirmation is received (cf. *UC3: Customer profile activation*)

- **Alternative scenarios:**

1. 3b. The chosen username already exists. HomeSys Cloud asks the Customer to choose a new username (step 2.)
2. 3c. The selected password is too short. HomeSys Cloud asks to choose a new password (step 2.)

1.3 *UC3*: Customer profile activation

- **Primary actor:** Customer
- **Interested parties:**
  - *HomeSys Cloud*: wants to validate the email of the Customer.
- **Preconditions:**
  - Customer has completed *UC2: Customer profile creation.*
- **Postconditions:**
  - The Customer profile is active (i.e., no longer flagged as 'pending').
- **Main scenario:**

1. The Customer receives the notification of a profile being created for them.
2. The Customer confirms the profile creation (e.g., by clicking a link).
3. The HomeSys Cloud asks the Customer if they want to register a Gateway (**Include:**. *UC4: Associate gateway to Customer*).

- **Alternative scenarios:**

1. 2b. The Customer did not confirm their profile creation within a certain time interval. The confirmation link becomes invalid and the user must follow the "forgot password procedure" to activate the profile. The registration of gateway will be initiated by the Customer at a later time.

1.4 *UC4*: Associate gateway to customer

- **Primary actor:** Customer
- **Interested parties:**
  - *Customer*: wants the devices they bought to synchronise data with HomeSys Cloud.
- **Preconditions:**
  - The Customer profile is already active in HomeSys Cloud (cf. *UC3: Customer profile activation*).
- **Postconditions:** The HomeSys Cloud is accepting data from the Gateway.
- **Main scenario:**

1. The Customer indicates they wants to associate a newly-installed HomeSys Gateway to the HomeSys Cloud.
2. The Customer enters the serial number of the gateway.
3. HomeSys Cloud validates that the provided serial number is valid (from a list of legit gateways) and not already associated to some other Customer.
4. The HomeSys Cloud associates the Gateway with a default configuration, depending on the Customer plan (e.g., gold, silver, bronze). The configuration will be sent to the Gateway the first time it connects to the Cloud (**Include:** *UC5: Send configuration to gateway*).
5. HomeSys notifies the Customer that the gateway is successfully registerd (**Include:** *UC11: Notify customer*).

- **Alternative scenarios:**

1. 4b. The entered serial number is invalid or links to an already associated device. HomeSys Cloud signals this to the Customer. Return to step 3.

1.5 *UC5*: Send configuration to gateway

- **Primary actors:** HomeSys Cloud
- **Interested parties:**
  - *Customer*: wants his devices to operate correctly and according to his defined profile.
- **Preconditions:**
  - The HomeSys Gateway is registered in HomeSys Cloud with a Customer.
  - Configuration data has changed, e.g., the initial configuration is to be sent, the Customer has changed their plan, a new alarm has been defined, etc.
  - The Gateway has made contact with the Cloud (as no configuration can be pushed without the Gateway contacting the Cloud first)
- **Postconditions:**
  - The HomeSys Gateway has received and stored the configuration data.
  - The HomeSys Gateway is synchronizing (and communicating) with the Cloud according to the Customer profile.
- **Main scenario:**

1. HomeSys Gateway contacts the Cloud, which replies with a command to download a new configuration
2. The HomeSys Gateway downloads the configuration data, stores it and processes it.
3. The HomeSys Gateway data sends a confirmation message to HomeSys Cloud.
4. HomeSys Cloud receives the confirmation message and records that the HomeSys Gateway has successfully received and processed the configuration data.
5. The Gateway runs the necessary reconfigurations

- **Alternative scenarios:**

1. 1b. After some time, the Gateway has still not received a command to download a new configuration (which has been ready for a while).
2. 1c. HomeSys Cloud notifies the HomeSys Call Center, which will inform the Customer
3. 3b. Something went wrong and the HomeSys Cloud did not receive a confirmation before the deadline. The HomeSys Cloud re-sends the configuration data: *Continue with step 1.*
4. 3c. After N attempts, HomeSys Cloud still did not receive a confirmation message from the device and notifies the HomeSys Call center.

1.6 *UC6*: Add a mote to the customer home

- **Primary actor:** Customer
- **Interested parties:**
  - *Customer*: wants to add a mote to enable the addition of new sensors/actuators.
- **Preconditions:** The HomeSys Gateway is powered up and is within physical reach for the mote.
- **Postconditions:**
  - HomeSys Gateway has stored the new mote information.
  - HomeSys Cloud has stored the new mote information.
- **Main scenario:**
  1. The MicroPnP mote broadcasts a heartbeat to the HomeSys Gateway.
  2. The HomeSys Gateway receives the heartbeat and stores the new mote information in the local database.
  3. The HomeSys Gateway includes the mote information upon the next connection to the Cloud.
  4. The HomeSys Cloud stores the mote information in the database.
- **Alternative scenarios:**
  1. 2b. The Gateway is too far from the mote to receive the data. The MicroPnP will keep on broadcasting the data until it can reach the gateway.

1.7 *UC7*: Plug in a sensore in a mote

- **Primary actor:** Customer
- **Interested parties:**
  - *Customer*: wants to add a new sensor to his house.
- **Preconditions:** There is a MicroPnP with an available open sensor plug-in port.
- **Postconditions:**
  - HomeSys Gateway has stored the new sensor information.
  - HomeSys Cloud has stored the new sensor information.
- **Main scenario:**
  1. The Customer plugs in the a sensor in the MicroPnP mote.
  2. The mote recognizes the sensor and sends a POST request to the HomeSys Gateway to notify that a new sensor has been plugged in.
  3. The HomeSys Gateway stores the sensor information in its database.
  4. The HomeSys Gateway sends the new sensor information to the Cloud.
  5. The HomeSys Cloud stores the mote information in the database.
- **Alternative scenarios:**
  1. 2b. The MicroPnP mote cannot recognize the mote and simply ignores it.

1.8 *UC8*: Send sensor data to the cloud

- **Primary actor:** HomeSys Gateway
- **Interested parties:**
  - *HomeSys Cloud*: wants to present sensor data to the paying Customers.
- **Preconditions:** The HomeSys Gateway has been registered and associated to the appropriate Customer.
- HomeSys Gateway has new sensor data in store for the Cloud.
- **Postconditions:**
  - HomeSys Cloud has stored and processed the sensor data.
- **Main scenario:**
  1. The HomeSys Gateway sends the latest sensor data (after a fixed time interval).
  2. The HomeSys Cloud confirms having received the transmission.
  3. The HomeSys Cloud looks up the Customer associated with the Gateway sending these sensor data.
  4. The HomeSys Cloud stores the measurements in the database.
- **Alternative scenarios:**
  1. 2b. The Gateway does not receive the confirmation from the Cloud and sends the data again
  2. 3b. The Gateway is not registered and the Cloud discards the sensor data

1.9 *UC9*: Rule (re)configuration

- **Primary actor:** Customer
- **Interested parties:**
  - *Customer*: wants to define event-handling rules. They might need to (i) define events of interest they would be notified about, (ii) link events to actuators actions/states, (iii) mark some events as alarming events.
- **Preconditions:** The Customer is logged in.
- **Postconditions:**
  - HomeSys Cloud has updated the customer profile.
  - HomeSys Cloud has prepared to reconfigure the Customer's HomeSys Gateway according to the customer profile.
- **Main scenario:**
  1. The Customer indicates they wants to configure the rules in their profile.
  2. HomeSys Cloud shows the current rules and details about available motes, sensors, actuators and the actions associated with specific actuators.
  3. The Customer provides the new rules for their profile and can delete/update existing rules.
  4. The HomeSys Cloud validates the Customer rules (e.g., to spot inconsistencies)
  5. HomeSys Cloud stores the new profile and confirms this to the Customer.
  6. HomeSys Cloud prepares to update the Gateway (**Include:** *UC5: Send configuration to gateway.*).

1.10 *UC10*: Evaluate rules on gateway

- **Primary actor:** HomeSys Gateway
- **Interested parties:**
  - *Customer*: wants to be notified of certain events/alarms in the system.
- **Preconditions:** HomeSys Gateway has received new sensor data.
- **Postconditions:** The Customer has received the notification.
- **Main scenario:**
  1. The HomeSys Gateway has processed some sensor data, which has activated a rule.
  2. The Gateway performs the actuations if necessary (e.g., sending an action to an actuator)
  3. The Gateway immediately sends an event or alarm to the Cloud.
  4. The HomeSys Cloud receives the event/alarm from the HomeSys Gateway.
  5. The event/alarm is stored (e.g., for the benefit of the HomeSys Call Center).
  6. The HomeSys Cloud notifies the Customer (**Include:** *UC11: Notify customer.*).

1.11 *UC11*: Notify customer

- **Primary actor:** HomeSys Cloud
- **Main scenario:**
  1. The HomeSys Cloud has to notify the Customer of an event or alarm.
  2. The HomeSys Cloud looks up for the communication channel to use for the Customer.
  3. HomeSys Cloud constructs the message for that specific communication channel.
  4. HomeSys Cloud sends the message to the Customer.
  5. The message is stored (e.g., for the benefit of the HomeSys Call Center).

1.12 *UC12*: View sensor data in the cloud

- **Primary actor:** Customer
- **Interested parties:**
  - *Customer*: wants to view the historical data about their home.
- **Preconditions:**
  - The Customer is logged in.
  - There is a HomeSys Gateway associated to the Customer.
- **Main scenario:**
  1. HomeSys Cloud shows the list of sensors monitored by HomeSys Gateway for the Customer.
  2. The Customer selects the sensor for which they wants to view historical data.
  3. The Customer chooses the desired period of consumption (within the limitations due to the subscription plan).
  4. HomeSys Cloud fetches the historical data and displays the data graphically.

1.13 *UC13*: View events and alarms in the cloud

- **Primary actor:** Customer
- **Interested parties:**
  - *Customer*: wants to view the historical data about their home.
- **Preconditions:**
  - The Customer is logged in.
  - There is a HomeSys Gateway associated to the Customer.
- **Main scenario:**
  1. The Customer requests the list of past events triggered by rules
  2. The HomeSys Cloud shows the list of notifications and alarms for the Customer so far (according to the service level agreement).

1.14 *UC14*: Changing of service level agreement

- **Primary actor:** Customer
- **Interested parties:**
  - *Customer*: wants to be able to choose a different level of service.
  - *HomSys Cloud*: wants to offer and sell additional services.
- **Preconditions:**
  - The Customer has created a profile.
- **Postconditions:**
  - The Gateway is updated.
- **Main scenario:**
  - The Customer wants to change their plan (e.g., to get premium services for a higher price)
  - HomeSys Cloud offers possible agreement scenarios to the Customer.
  - The Customer chooses and agrees on specific plan.
  - The HomeSys Cloud updates the Customer profile
  - The HomeSys Cloud logs the necessary info in order to be able to produce a correct and fair invoice at the end of the payment period.

1.15 *UC15*: Unregister customer

- **Primary actor:** Customer
- **Interested parties:**
  - *Customer*: does not want to use HomeSys anymore.
- **Preconditions:**
  - The Customer was registered to HomeSys Cloud.
- **Postconditions:**

- The Customer is unregistered from HomeSys Cloud. His profile is marked as inactive.
- HomeSys Cloud does not accept anymore the data sent by the Gateway.
- The final invoice is generated and the Customer is not charged anymore.

- **Main scenario:**

1. The Customer indicates they want to start the unregistration process.
2. HomeSys Cloud deactivates the Customer profile and unregisters their Gateway.
3. HomeSys Cloud removes all associated sensor data from the database.

1.16 *UC16*: Customer invoicing

- **Primary actor:** HomeSys Cloud
- **Interested parties:**
  - *Customer*: want to be charged with the correct amount.
- **Preconditions:**
  - The scheduled monthly invoicing time has come.
- **Postconditions:**
  - The Customer has received an e-invoice.
- **Main scenario:**

1. The HomeSys Cloud computes the charge for the Customer, according to the Customer plan and the potential plan change the Customer has made during the invoicing period.
2. A notification is sent to the user with the payment details (amount, account, OCR) and a breakdown of the costs.

2 Quality requirements

The quality requirements are documented in the form of *quality attribute scenarios*.

2.1 *QAS1*: HomeSys Cloud sensor data DB failure

The internal database responsible for storing measurements fails/crashes.

- **Source:** Internal
- **Stimulus:** The database in HomeSys Cloud responsible for storing all sensor data fails or crashes.
- **Artifact:** Internal subsystem
- **Environment:** Normal execution
- **Response:**
  - The failure does not affect the availability of other types of persistent data, such as the customer profiles, configuration data, invoicing data, etc.
  - Detection:
    - HomeSys Cloud is able to detect a problem with the database and goes into a maintenance mode.
    - The System Administrator is notified of this problem.
  - Resolution:
    - Once the database has crashed or failed the HomeSys Cloud switches to a replica of the data automatically (no impact on the service continuity).
    - The administrator replaces the failing database component.
    - The replacement is synchronized with the replica.
    - The system returns to normal mode.
- **Response measure:**
  - A fully synced replica is available 100% ot the time on stand-by in a physically different location.
  - Detection of failures happens within 5 seconds.
  - The System Administrator are notified within 1 minute.
  - After a failure, the normal mode is restored within 2 hours.
  - During maintenance mode, no new measurements are lost.

2.2 *QAS2*: HomeSys Gateway not sending data to the Cloud provider

The Cloud provider stopped receiving data from a registered HomeSys Gateway.

- **Source:** Internal or External
- **Stimulus:**
  - The external communication channel between the HomeSys Gateway and the HomeSys Cloud operators is failing,
  - HomeSys Gateway has failed, or
  - an internal communication component in HomeSys Gateway has failed
- **Artifact:** external device, external communication channel(s) or internal communication (sub)System(s)

- **Environment:** Normal mode
- **Response:**
  - Detection:
    * The HomeSys Cloud services is able to detect that the Gateway is not sending data anymore.
    * The HomeSys Gateway should be able to detect the failure of an internal component related to the communication with the Gateways.
  - Resolution:
    * The HomeSys Cloud should log when sensor data from a Gateway did not arrive.
    * After a grace period (which depends on the subscription plan of the Customer), the HomeSys Cloud notifies the HomeSysCall Center about the problem.
    * The Call Center contacts the customer during business hours to solve the problem.
    * If the Gateways comes back online, the Call Center is notified and the problem is flagged as solved.
- **Response measure:**
  - The detection happens within 5 seconds after the grace period has expired.
  - The HomeSys Call Center is notified within 1 minute.

2.3 *QAS3*: Timely notification of alarms

Notification of the Customer (and possibly HomeSys Call Center) should be sent quickly and according to given priorities.

- **Source:** E.g., smoke or humidity sensor / HomeSys Gateway
- **Stimulus:** The sensor data trigger a rule, resulting in an alarm.
- **Artifact:** System
- **Environment:** Normal mode
- **Response:**
  - The notification of alarms must be prioritized over the processing of sensor data and the notification of other (non-alarm) events.
  - Certain alarms (e.g., smoke) have priority over other alarms (e.g., high humidity) and the priorities are set centrally (in a system-wide manner) in the Cloud. The priorities of alarms are defined as green, yellow and red. The assignment of alarms to priorities and can be altered.
- **Response measure:**
  - All notifications of alarms must be sent to the Customer within 10 seconds.

2.4 *QAS4*: Large number of Customers

HomeSys should be hable to handle a successful business with several Gateways being sold.

- **Source:** Customer/Gateway
- **Stimulus:** A Customer has accessed their profile on the Cloud, or a Customer's Gateway has sent sensor data
- **Artifact:** The HomeSys Cloud

- **Environment:** Normal mode
- **Response:**
  - The HomeSys Cloud replies to the service requests of the Customers, or processes the data from the Gateways.
- **Response measure:**
  - HomeSys should be able to able to deal with 100,000 Gateways and, accordingly, to handle a large number of clients connected to the Customers.

2.5 *QAS5*: Authentication of Cloud users

Any user-facing functionality of HomeSys should be only accessible to users with proper credentials (Customer, System Administrator, or Call Center Operator).

- **Source:** Customer/System Administrator/Call Center Operator
- **Stimulus:** A Customer accesses their profile, or any other functionality of the Cloud (same for other user types)
- **Artifact:** The HomeSys Cloud
- **Environment:** Normal mode
- **Response:**
  - At the beginning of each session, the user has to log in with credentials (e.g., email and password) and obtains a session ID (which serves the purpose of authentication token).
  - The HomeSys Cloud checks that the user is using a valid and secure session ID.
- **Response measure:**
  - No functionality is accessible without a valid authentication token.
  - The authentication token is hard to guess.

2.6 *QAS6*: Users are accountable for their actions

Any action initiated by a user should be logged securely.

- **Source:** Customer/System Administrator/Call Center Operator
- **Stimulus:** A Customer accesses their profile, or any other functionality of the Cloud (same for other user types)
- **Artifact:** The HomeSys Cloud
- **Environment:** Normal mode
- **Response:**
  - Before servicing any request, the request parameters (including the service name), the customer identity, the time, and other contextual information are logged in a secure DB.
- **Response measure:**
  - No functionality is accessible without a valid trace.

E.3 HomeSys Architectural Description

Home Monitoring System (HomeSys) - Architectural description

DAT220/DIT554 - Advanced Software Architecture

2016

Abstract

HomeSys is used for remote monitoring of residential apartments and houses. The main goal of the system is to provide the customers with the necessary tools to monitor their home and to get prompt notifications upon critical events (e.g., smoke, low / high temperature, etc.).

This document describes an architectural design for the Home Monitoring System (HomeSys). The final architecture has been designed based on the domain analysis, the requirements and the priorities of these requirements.

Contents

1 Context diagram **2**

2 Gateway component diagram **2**

3 Cloud component diagram **3**

3.1 Operator Facade decomposition 4

3.2 Customer Facade decomposition 4

3.3 Gateway Facade decomposition 6

3.4 Database decomposition 7

4 HomeSys deployment diagram **7**

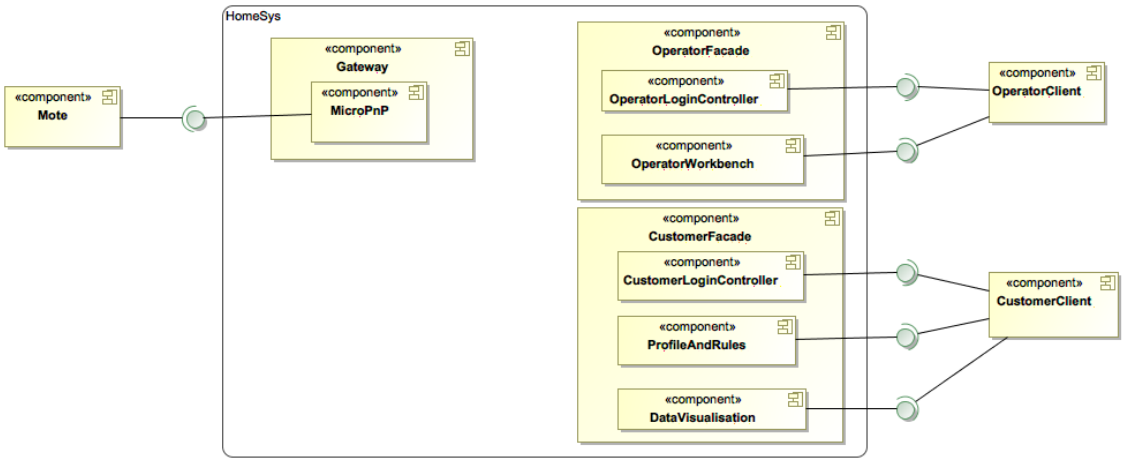


Figure 1: HomeSys context diagram

1 Context diagram

The context diagram of the HomeSys is given in Figure 1. As shown, three distinct external entities communicate with the system:

- **The mote** is a hub where the sensors and actuators are plugged onto and installed throughout the house. The mote sends sensor data to the gateway and can get actuations from the gateway.
- **The operators** use the appropriate operator interface, take on every sort of question about the system from the customer's side and help them tackle any issues with the installation and configuration.
- The HomeSys Cloud services are accessible to **the customer** via the browser or a mobile app. These services allow the user to manage the HomeSys as well as view all the historical sensor data on the information dashboard.

2 Gateway component diagram

The HomeSys gateway is a hardware device that enables smart home automation. It relays the sensor data to the cloud and manages the actuators. The decomposition of the *Gateway* component is presented in Figure 2. As shown, the *Gateway* component is decomposed into six subcomponents: the *RuleEngine*, the *GatewayDB*, the *MicroPnP* component, the *GatewayWatchDog*, the *PeriodicScheduler* and the *CloudComm*.

RuleEngine is responsible for managing defined rules. The *RuleEngine* stores all the rules in the *GatewayDB*. The new rules that are added by the customer on the cloud are synced to the *RuleEngine* via the *CloudComm*. The *GatewayDB* is used by the *MicroPnP* to store the newly received sensor data. The *MicroPnP* also notifies the *RuleEngine* to evaluate the rules when new data is received. If required by the rules the *RuleEngine* will invoke the actuators via the *MicroPnP* and trigger alarms and/or events via the *CloudComm*.

CloudComm is responsible for the communication between the Cloud services and the Gateway. The *CloudComm* will periodically communicate to the Cloud to send sensor data (fetched from the *GatewayDB*) and fetch new user configuration. The configuration can be new rules that are synced

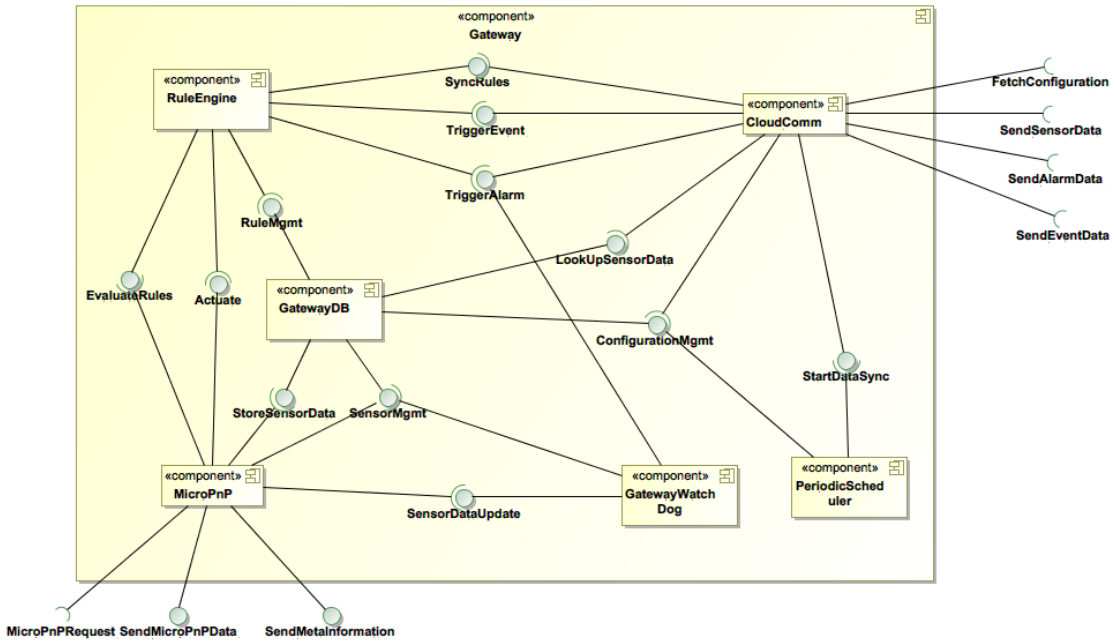


Figure 2: HomeSys Gateway

with the *RuleEngine* or SLA stored in the *GatewayDB*. The *CloudComm* is triggered by the *PeriodicScheduler* component to start the Cloud synchronisation as well as by the *RuleEngine* to send any alarm or event data.

GatewayWatchDog is used to detect if sensors are working properly. This component will retrieve all information from the *GatewayDB* upon initial startup and will receive periodic sensor heartbeat updates. If a heartbeat is missing this component will trigger an alarm by leveraging the *CloudComm*.

No further decomposition is provided for the Gateway subcomponents.

3 Cloud component diagram

The HomeSys cloud is a software system including a portal where the customers can log in and view the information about their houses.

The decomposition of the *Cloud* component is presented in Figure 3. As shown, the Cloud component is decomposed into five subcomponents: the *GatewayFacade*, the *NotificationController*, the *OperatorFacade*, the *CustomerFacade* and the *Database*.

GatewayFacade is responsible for the communication between the gateway and the Cloud. It receives sensor, event and alarm data from the Gateway and allows the Gateway to fetch the new configuration data. The decomposition of the *GatewayFacade* component is presented in Section 3.3.

NotificationController sends notifications to customers via different communication channels (i.e., email, SMS or mobile app push) based on the configuration stored for the specific customer in the *Database*.

OperatorFacade authenticates operators and allows them to manage practically everything to support the customers. The decomposition of the *OperatorFacade* component is presented in Section 3.1.

CustomerFacade authenticates customers and allows them to manage their profile, their HomeSys settings and view the own data. The decomposition of the *CustomerFacade* component is presented

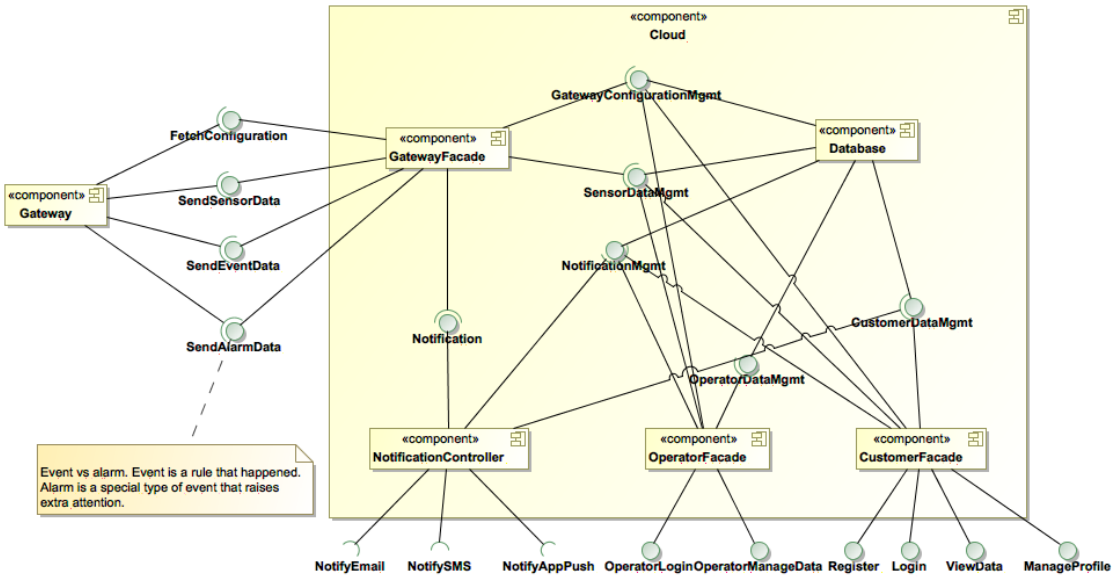


Figure 3: HomeSys Cloud

in Section 3.2.

Database component stores all relevant data. This component is further decomposed (see 3.4).

3.1 Operator Facade decomposition

OperatorFacade authenticates operators and allows them to support customers by managing their data. The decomposition of the *OperatorFacade* is presented in Figure 4. As shown, the *OperatorFacade* component is decomposed into three subcomponents: the *OperatorLoginController*, the *OperatorWorkbench* and the *AccountabilityController*.

OperatorLoginController is responsible for authentication of operators (via username and password). This component will also validate the credentials (e.g., using a session ID) upon each request handled by the *OperatorWorkbench*.

OperatorWorkbench provides operators with the necessary toolset to manage all customer data. Upon every request to this component it will validate the operator credentials by using the *OperatorLoginController*.

AccountabilityController records and stores all actions of operators.

3.2 Customer Facade decomposition

The *CustomerFacade* authenticates customers and allows them to manage their own profile and data. The decomposition of the *CustomerFacade* is presented in Figure 5. As shown, the *CustomerFacade* component is decomposed into four subcomponents: the *AccountabillityControler*, the *CustomerLoginController*, the *DataVisualisation* and the *ProfileAndRules*.

CustomerLoginController is responsible for the registration of customers as well as their authentication (via email and password). This controller will be always invoked (by the DataVisualization and the ProfileAndRules) to validate the credentials of the customer (e.g., by checking a session ID).

DataVisualisation is used for the presentation of data on various customer devices. This component will use the *CustomerLoginController* to validate the customer credentials.

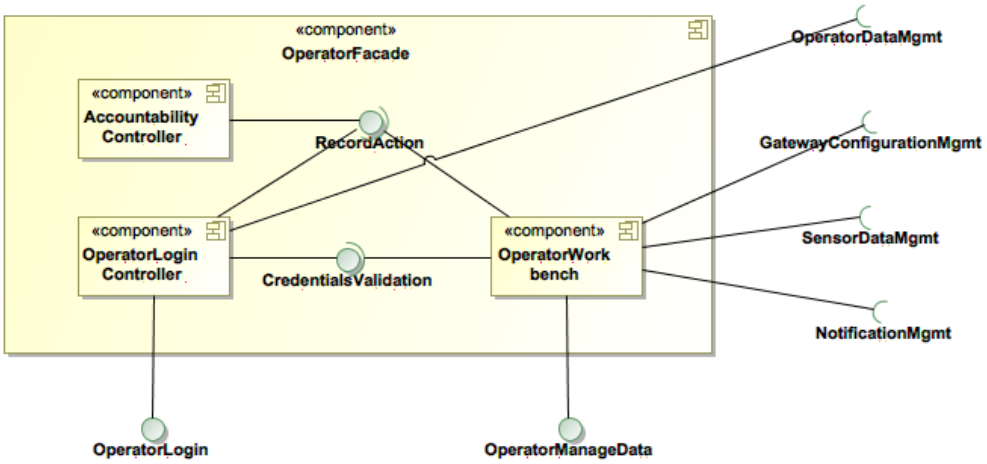


Figure 4: Operator Facade

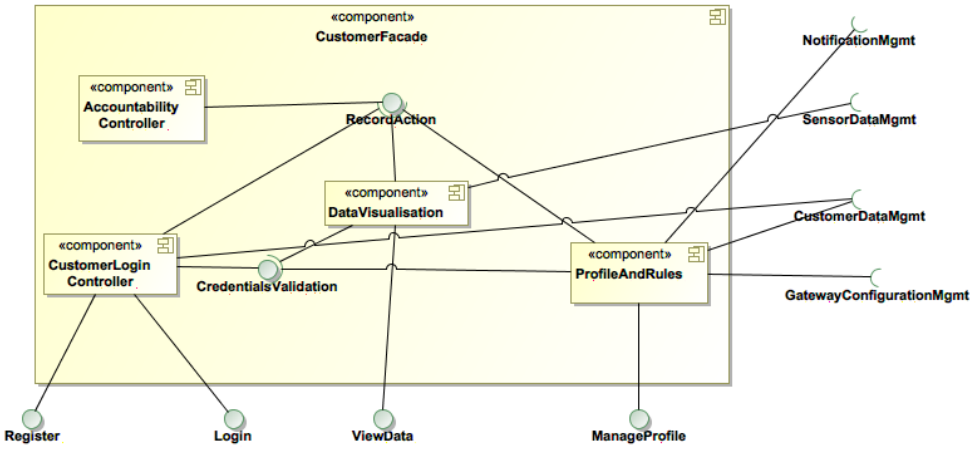


Figure 5: Customer Facade

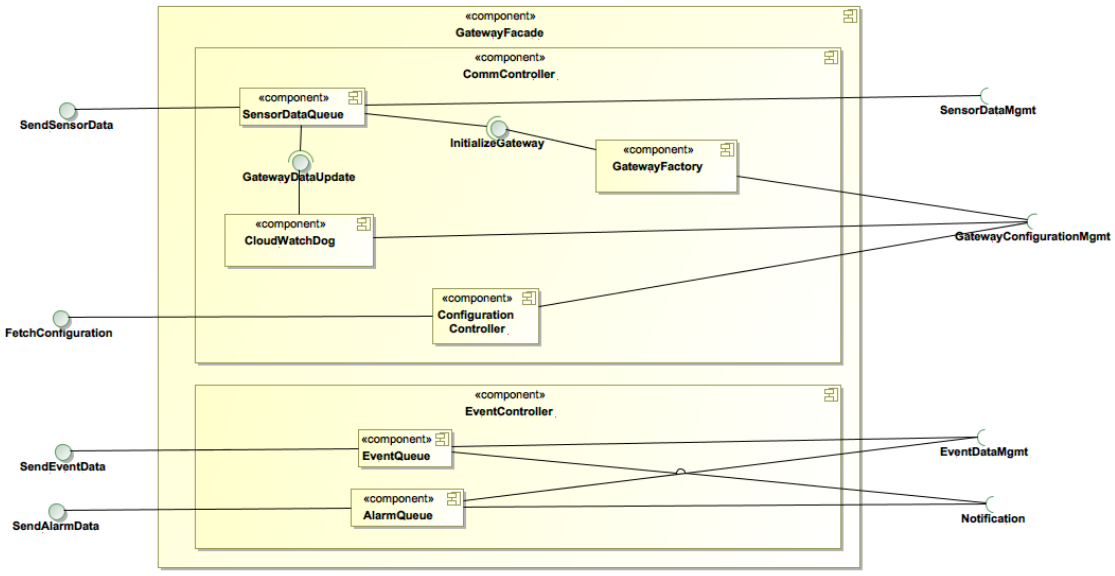


Figure 6: Gateway facade

The *ProfileAndRules* provides customers with the necessary tools to manage their own data, settings, rules, etc. This component will use the *CustomerLoginController* to validate the customer credentials.

The *AccountabillityControler* records and stores all actions performed by the customers.

3.3 Gateway Facade decomposition

The *GatewayFacade* is responsible for communication between the gateway and the cloud. It receives sensor, event and alarm data from the Gateway and provides the Gateway with configuration data. The decomposition of the *GatewayFacade* is presented on figure 6. The *GatewayFacade* component is decomposed into six subcomponents: the *SensorDataQueue*, the *GatewayFactory*, the *CloudWatchDog*, the *ConfigurationController*, the *EventQueue* and the *AlarmQueue*.

The *SensorDataQueue* processes all the sensor data received from the gateway. This component will (upon each sensor data connection) notify the *CloudWatchDog* that the gateway is alive. It will also provide the *GatewayFactory* with information on new gateway connections that are still unknown to the cloud.

The *GatewayFactory* is responsible for initialisation of the new HomeSys Gateways. The initialization means that once a new gateway starts sending data it will be automatically added to the database.

CloudWatchDog is notified every time a gateway has made a connection. This component will fetch from the database the expected connection intervals and it will trigger an alarm when too many connections have gone missing.

ConfigurationController enables the gateways to get the new configuration parameters. This component simply fetches the configuration from the database and provides it to the HomeSys gateway. Note that a confirmation is sent by the HomeSys gateway to confirm the receipt of the new configuration.

EventQueue processes all events received from the gateway and stores them in the database.

AlarmQueue is similar to the *EventQueue*, however it is operating with strict deadlines in order to notify the customers promptly.

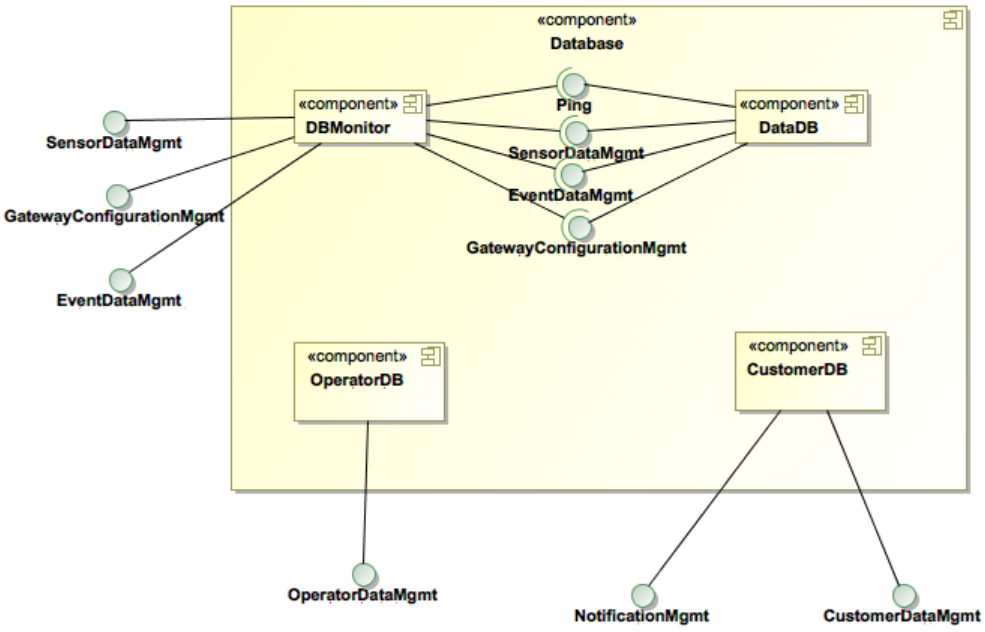


Figure 7: Database

3.4 Database decomposition

The *Database* component stores all relevant data. The decomposition of the *Database* component is presented in Figure 7. As shown, the *Database* component is decomposed into four subcomponents: the *DBMonitor*, the *DataDB*, the *OperatorDB* and the *CustomerDB*.

The *DBMonitor* is used as a replication manager for the *DataDB* component, which contains sensor data, events, alarms and gateway configurations. The idea is that the *DBMonitor* will regularly ping the *DataDB* and if it fails the *DBMonitor* will immediately switch to the replica database (see also Section 4).

OperatorDB component stores all the data that is relevant to operators.

CustomerDB component stores all the customer related data.

4 HomeSys deployment diagram

The Deployment diagram is shown in Figure 8. The components are deployed in three groups:

1. **Front-end nodes** hosting facades and supporting components. As shown in bottom part of the figure, two nodes are connected to the outside world: the *Operator Facade Node* and the *Customer Facades Node*.

2. **Business nodes** hosting the business logic. As shown in the left part of the figure, this group includes: several *Gateway Nodes* (the physical gateway devices) and the *Gateway Communication Node* (dealing with the communication with gateways from the cloud side).

3. **Back-end nodes** hosting the data. As shown in the right part of the figure, this group includes: the *Database Monitoring Node* (used for pushing and pulling data to and from database), the *Primary Database Node* and the *Secondary Database Node*.

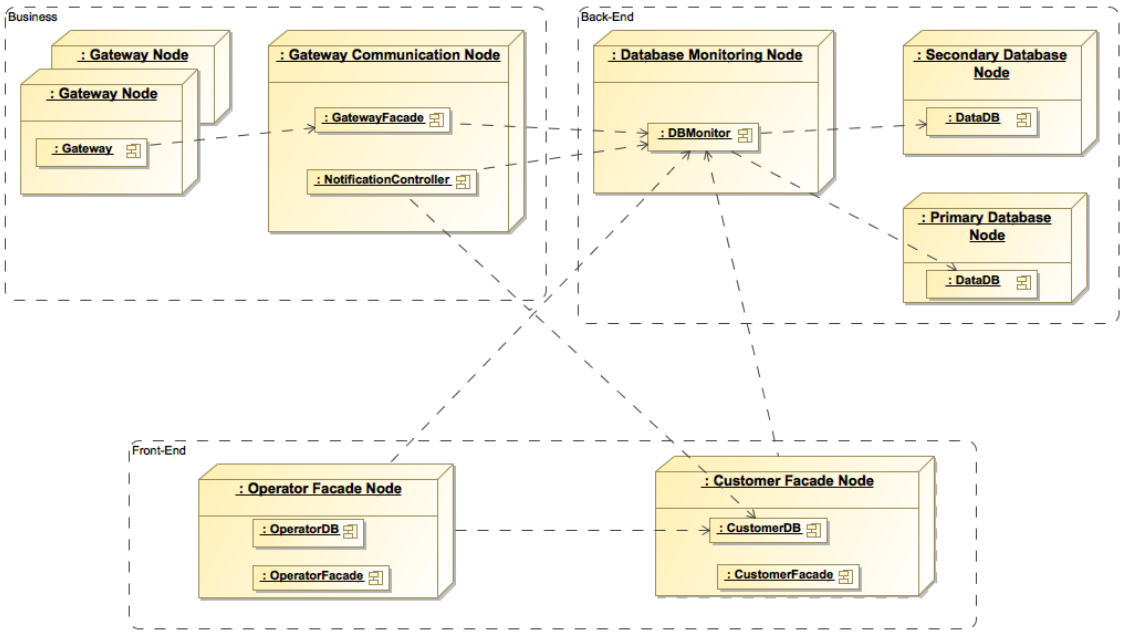


Figure 8: HomeSys deployment diagram