Project title: Enforceable Security in the Cloud to Uphold Data Ownership
Project acronym: ESCUDO-CLOUD
Funding scheme: H2020-ICT-2014
Topic: ICT-07-2014
Project duration: January 2015 – December 2017

D2.4
Report on Requirement-Based Threat Analysis

Editors: Ahmed Taha (TUD)
Patrick Metzler (TUD)
Neeraj Suri (TUD)
Reviewers: Giovanni Livraga (UNIMI)
Stefano Paraboschi (UNIBG)

Abstract

D2.4 reports on T2.4 activities to explore the potential of conducting threat analysis based on requirements. As ESCUDO-CLOUD is driven by its Use Cases (UCs), D2.4 first extracts user data ownership related requirements from the four UCs. Subsequently the requirements are analyzed, individually and collectively, to identify the assumptions (and consequences of assumption violation) underlying them. The intent is to develop a dependency assessment technique supporting this analysis, as well as means to automate threat analysis. D2.4 establishes the viability of determining threats from requirement analysis, and progressively builds upon W2.3 that was released at M15.

Type	Identifier	Dissemination	Date
Deliverable	D2.4	Public	2017.03.31

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 644579. This work was supported in part by the Swiss State Secretariat for Education, Research and Innovation (SERI) under contract No 150087. The opinions expressed and arguments employed herein do not necessarily reflect the official views of the European Commission or the Swiss Government.

ESCUDO-CLOUD Consortium

1.	Università degli Studi di Milano	UNIMI	Italy
2.	British Telecom	BT	United Kingdom
3.	EMC Corporation	EMC	Ireland
4.	IBM Research GmbH	IBM	Switzerland
5.	SAP SE	SAP	Germany
6.	Technische Universität Darmstadt	TUD	Germany
7.	Università degli Studi di Bergamo	UNIBG	Italy
8.	Wellness Telecom	WT	Spain

Versions

Version	Date	Description
0.1	2017.03.07	Initial Release
0.2	2017.03.27	Second Release
1.0	2017.03.31	Final Release

List of Contributors

This document contains contributions from different ESCUDO-CLOUD partners. Contributors for the chapters of this deliverable are presented in the following table.

Chapter	Author(s)
Executive Summary	Ahmed Taha (TUD), Patrick Metzler (TUD), Neeraj Suri (TUD)
Chapter 1: Introduction	Ahmed Taha (TUD), Patrick Metzler (TUD), Zhazira Oskenbayeva (TUD), Neeraj Suri (TUD)
Chapter 2: Requirements Analysis from Use Case 1	Christian Cachin (IBM)
Chapter 3: Requirements Analysis from Use Case 2	Florian Kerschbaum (SAP)
Chapter 4: Requirements Analysis from Use Case 3	Ali Sajjad (BT)
Chapter 5: Requirements Analysis from Use Case 4	Mercedes Castano Torres (WT)
Chapter 6: Dependency Analysis Across Requirements	Ahmed Taha (TUD), Patrick Metzler (TUD), Neeraj Suri (TUD)
Chapter 7: Summary Analysis from Use Cases and Threat Analysis	Andrew Byrne (EMC), Silvio La Porta (EMC)
Chapter 8: Automation Techniques for Threat Analysis	Andreas Fischer (SAP), Daniel Bernau (SAP), Anselme Kemgne Tueno (SAP)
Chapter 9: Conclusions	Ahmed Taha (TUD), Patrick Metzler (TUD), Neeraj Suri (TUD)

Contents

List of Figures

List of Tables

Executive Summary

As a technique, threat analysis aims to uncover threats that can compromise the services expected from a target system. Typically, threat analysis is conducted at a variety of levels spanning varied levels of code access (e.g., whitebox, graybox, blackbox), at the level of interface specifications, at the protocol or functional level, and similar variations. As ESCUDO-CLOUD is driven by its Use Cases (UCs) where the core functionality is specified by the UC requirements, hence D2.4 investigates if a UC level requirements analysis can be utilized to conduct threat analysis. Consequently, D2.4 focuses on threats whose impact results in violation of data ownership principles as related to the ESCUDO-CLOUD Use Cases (UCs).

To systematically understand and prioritize the threats, it is first essential to capture the specific data owner requirements on functionality, performance and security, with respect to the specific data to be outsourced. Hence the assumptions behind the UC requirements related to data ownership are identified. These include both direct assumptions as well as indirect assumptions that were implicitly assumed but not explicitly specified. For each of these assumptions, the next step is to determine the implication of their violation. The violations are assessed by their type, their impact and also the likelihood of their occurrence. This process of requirements analysis was conducted individually (per UC) and collectively (across the UCs), to identify the range of assumptions and consequences of assumption violation underlying them. The task (termed as *"Requirements based threat analysis (RBTA)"*) also provided an input for estimating the risks based on such requirement analysis of the data owner.

The manual RBTA process was able to establish the viability of identifying threats based on a UC requirements analysis. At the same time, the manual process is both time intensive and makes no assurance on completeness of the analysis. Thus, building on the basic viability of RBTA result, the deliverable next developed a systematic analytic technique that enumerates the set of UC requirements and then determines all possible direct/indirect dependencies across them to conduct a generalized threat analysis from their requirements. The approach was validated for its effectiveness for its effectiveness on the ESCUDO-CLOUD UCs, and is also generalized to apply to other UCs based on the availability of their requirements. As the type of identifiable threats depend on the level of detail available in the requirements, the deliverable also considers the broader issue of generalized threat analysis and presents an approach to automate threat analysis.

D2.4 progressively builds on M15/W2.3 and reports on the T2.4 activities that explore the potential of conducting threat analysis based on requirements. Overall, D2.4 was able to establish both the potential and utility for ESCUDO-CLOUD UC level requirements based threat analysis, and also develop rigorous techniques to support RBTA.

1. Introduction

Cloud infrastructures are subject to multiple threats, which may target the technological foundation of the Cloud infrastructure (e.g., virtualized environment) or architectural aspects of the next generation of Cloud storage system (e.g., availability). Virtualization is the enabling technology for the different Cloud computing models and is expected to offer secure logical isolation between virtual machines. Recently, researchers have demonstrated that the isolation property may be violated by extracting the private key of co-located users or by exchanging secrets between colluding users [ZJRR12]. Different factors affect the exploitability of these threats, such as the noise level of side-channels, or deployed mechanisms. Similarly, the perceived lack of trust and control over outsourced data raises major concerns about the security and privacy of the outsourced data. To alleviate these concerns, a systematic approach is required in order to estimate the feasibility of exploitation of these threats in a specific operational context and under the stipulated assumptions. Additionally, these threats need to be analyzed with regards to the requirements of the data owner in terms of functionality, performance and security, to the aim of properly identifying which data to be outsourced are critical and accordingly prioritize threat modeling activities.

The primary specification for the ESCUDO-CLOUD Use Cases are their requirements. Thus, T2.4 explores if the Use Case requirements can form the basis for uncovering security threats for the targeted property of Data Ownership. In order to accomplish this, four sub-problems need to be addressed. The first one is to outline the assumptions (both direct and indirect) behind the requirements to ascertain if the given process can provide meaningful information to identify potential threats along with a corresponding threat severity assessment. A preliminary analysis showed the viability of outlining the assumptions at the level of individual requirements in all the Use Cases. On this positive result, the second target is to assess risks associated with each threat and strategy to mitigate these risks. The third (and key contribution) target is to develop a novel methodology that can systematically enumerate the dependencies across the service requirements to reveal potential threats. Finally, the fourth target is to provide automation for threat analysis.

Threat analysis (TA) is a process to systematically identify, detect, and evaluate security vulnerabilities. Threat analysis considers the full spectrum of threats for a given system. For example, TA can be conducted at the level of architecture diagrams, at trust boundaries, over detailed data flow diagrams, attack surfaces at the component or functional levels, on source code, via misuse cases and many other dimensions. In this document, we consider requirement based threat analysis to identify threats, whose impact results in violation of security requirements as related to the ESCUDO-CLOUD Use Cases.

1.1 Interactions with other Work Packages

The work discussed in this document is related to and interacts with the other ESCUDO-CLOUD work packages as follows (Figure 1.1):

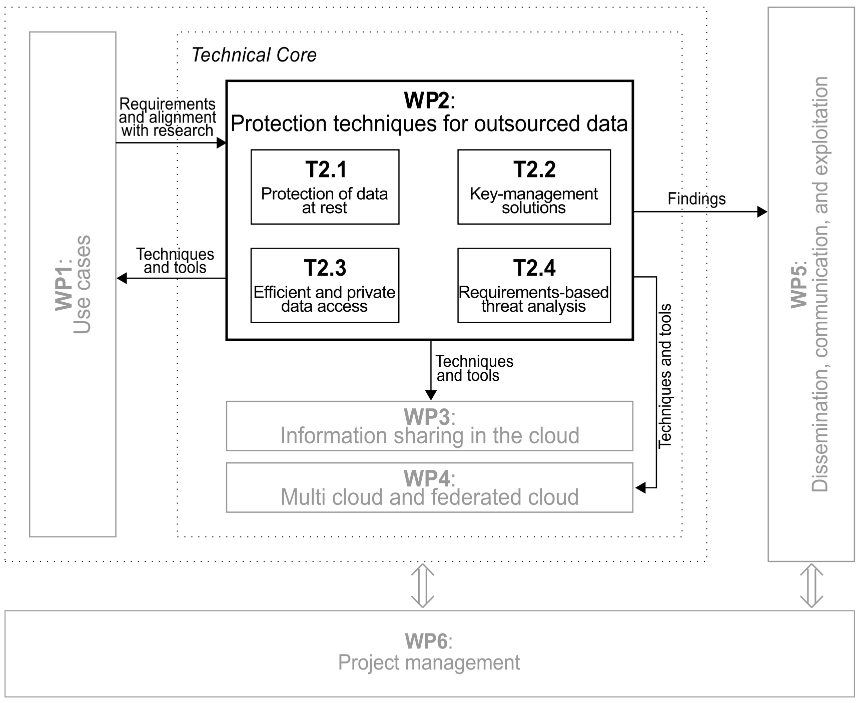

Technical Core
WP2: Protection techniques for outsourced data
T2.1 Protection of data at rest
T2.2 Key-management solutions
T2.3 Efficient and private data access
T2.4 Requirements-based threat analysis
Requirements and alignment with research
Techniques and tools
Findings
Techniques and tools
Techniques and tools
WP1: Use cases
WP3: Information sharing in the cloud
WP4: Multi cloud and federated cloud
WP5: Dissemination, communication, and exploitation
WP6: Project management

Figure 1.1: Interactions of T2.4/D2.4 with other WP tasks of ESCUDO-CLOUD

- **Work Package 1** provides the UCs for ESCUDO-CLOUD. The UCs and related requirements from deliverables D1.1 and D1.2 form the basis for the threat analysis. In turn, the techniques discussed in this report will be applied to the UCs as part of Work Packages 1, 3 and 4.
- **Work Package 3** receives the threat analysis techniques that supports the implementation of a Cloud integrity/consistency verification tool.
- **Work Package 4** receives the dependency assessment technique for the Multi-Cloud model evaluation and dependency assessment.
- **Work Package 5** disseminates and exploits the threat analysis and dependency assessment techniques for Cloud storage provided in this document.

T2.4 utilizes the ESCUDO-CLOUD Use Case requirements provided from D1.1/1.2. These requirements provide abstract level information without revealing low-level details of the Use Cases. Thus, the requirements from D1.1/1.2 constitute the reference level of details to drive requirement based threat analysis. Naturally, the requirement based threat analysis yields results only to the level of details provided by the requirements. Consequently, there is no claim of completeness of implementation level violations from the threat analysis. Instead threat analysis is an exploration to determine the potential of ascertaining design-level threats from the analysis of requirements. The completeness of a requirement based threat analysis inherently depends on the level of detail of the provided requirements.

1.2 Threat Analysis (TA) Approach

ESCUDO-CLOUD is predominantly driven by the enumeration of requirements from actual Use Cases with the resultant articulation of user data ownership scenarios. Thus, ESCUDO-CLOUD explores TA at the requirements level. Specifically, the TA focuses on threats that can potentially violate the user's data ownership requirements of security, functionality and performance. The UCs also form the basis for testing/validation. Each Use Case addresses a specific application scenario, which can be classified as Infrastructure Provisioning, Cloud Storage, and Cloud Processing scenarios.
Use Case 1 (UC 1): This Use Case relates to a Cloud-storage platform, which supports server-side encryption with flexible key-management solutions, to be used with OpenStack framework. This facilitates user data security in the OpenStack framework.
Use Case 2 (UC 2): This Use Case covers secure enterprise data management in the Cloud, specifically secure data sharing among the business parties. The goal of this Use Case is to provide solutions that allow a data owner to share information without losing control over his data.
Use Case 3 (UC 3): This Use Case considers the application of data protection as a service via a Cloud service store that enables customers to protect their data stored on multi-Cloud environments including federated secure Cloud storage.
Use Case 4 (UC 4): This Use Case considers Cloud service brokers or intermediaries offering a secure Cloud data storage capability to their customers while possibly leveraging other Cloud providers for storing data and ensuring that data is protected from both other Cloud providers and users.

Use Case Requirements based TA Process

For each Use Case, we identify security requirements from multiple dimensions as depicted in Figure 1.2. These dimensions are a) security properties (Confidentiality, Integrity and Availability); b) sharing requirements; c) access requirements. For example, security requirements for Use Case 1 are CRUD (Create, Read, Update, and Delete) operations for infrastructure and tenants' keys and scalable design of key management system. We also identify the relationship between requirements and their impact on performance, functionality and security properties. Afterwards, a priority level is assigned to each requirement to ascertain the dependencies among the requirements e.g., scalable design of key management depends on the CRUD operations for infrastructure and tenants' keys. This helps to better understand the threats associated to each requirement and the consequence of its violation on other requirement(s).
Once we have identified the basic requirements related to each Use Case and their dependencies, the next step is to ascertain the assumptions behind the requirements, which can be direct and indirect. For example, the direct assumption for CRUD operations for tenant keys is the need of a trusted administrator and the indirect assumption is the need fors a trusted infrastructure. If such assumptions are violated, it will result in confidentiality violation for the tenant data which is a high severity threat.
The final step is to model security requirements and their dependencies utilizing a proposed hierarchical/tree structure. To summarize threats based on requirements, we find common requirements across all UCs and categorize them. One such category is "data confidentiality through access control policies", which is required by all UCs. We then estimate the maximum likelihood of its violation along with the severity of the threat in case of violation(s).

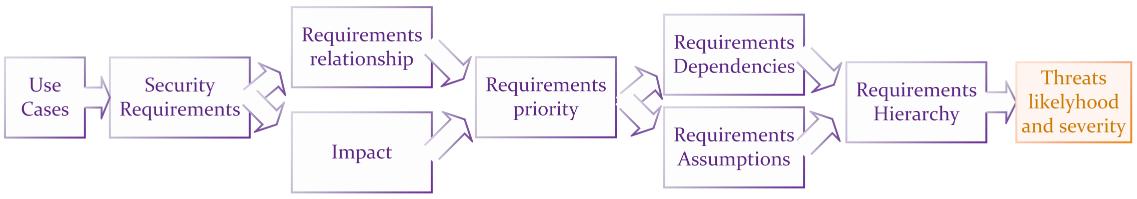


Figure 1.2: Threat analysis process

1.3 ESCUDO-CLOUD Innovation

The main innovation in T2.4 is the ongoing development of a requirements-driven methodological process that ascertains direct and indirect dependencies across the requirements to highlight inconsistent specifications or assumptions that can lead to security breaches. Each Cloud service is presented as a tree (Boolean or X-tree as relevant) that outlines the hierarchy of requirements providing the Cloud service. As multiple services are elaborated, the methodology explores the linkages and dependencies across all the service requirements to ascertain either their compatibility or the incompatibility - the latter identifying the vulnerabilities. The linkages are determined as vertical dependencies (across levels, either within or across services) and horizontal (across the same level within and across services) and also for both direct and indirect dependencies. This inter-linked services model is a new contribution to the area of threat modeling and has resulted in a systematic 3-step methodology (Initial design of the Dependncy Model, Validation of the Model, Use Case requirements reorganization) and was published in [TMT$^+$16]. The proposed methodology allows for rigorizing the compatibility searches across the requirements. A related highlight is the validation of the proposed RBTA that is being conducted on all four Use Cases along with a cross-analysis across the Use Cases to ascertain common requirements across them.

As mentioned earlier, a prominent limitation of the current threat modeling approaches is that they require significant manual work to identify and assess potential threats. Additionally, the analysis often lacks focus on the primary assets that require protection. As an innovation, T2.4 develops an efficient architectural threat modeling approach supported by a tool that allows partly automating and focusing security assessment based on (1) assets and security objectives in the system and (2) high-level design enriched with security aspects. By complementing architecture diagrams with security-related information, a lightweight (automated) threat identification is achieved. Besides, the information captured during this analysis can serve as a good high-level documentation of the threat modeling results.

The overall requirements based threat analysis approach is still under development, and the work is extending the methodology proposed by us in [TMT$^+$16] for application to all Use Cases.

1.4 Outline

The rest of this document is structured as follows:

Chapters 2-5 provide an overview of the security requirements of each individual Use Case, the assumptions behind their requirements, and the likelihood of these assumptions being violated. This forms the essential basis for threat/attack tree modeling developed in Chapter 6.

Chapter 6 analyzes dependencies across different Use Case requirements. This analysis ensures that the threat modeling captures relationships between requirements and how a violation in one requirement affects other requirement(s). Furthermore, this chapter presents a dependency representation model for validating each Use Case requirements by checking the existence of conflicts that occur due to different dependency relations across requirements. The process of analyzing conflicts is both iterative and interactive.

Chapter 7 cross examines all ESCUDO-CLOUD Use Case requirements to group the common requirements together into logical categories. Consequently, using these categories to present the Use Cases threat analysis, it depicts the Use Cases distribution on the axes of *Violation Likelihood* and *Threat Severity*, as well as identifying in which Use Case the highest threat concerns exist.

Chapter 8 develops a partly automated technique for threat/attack tree modeling based on the system security objectives and a high-level design enriched with security aspects.

Chapter 9 summarizes the developments of this deliverable.

2. Requirements Analysis from Use Case 1

This use case considers storage encryption and key-management techniques for Cloud platforms, using the example of OpenStack. Key management for the Cloud requires different solutions from traditional deployments, where control and security mostly rely on physical control and ownership of devices. In the Cloud model, it is nearly impossible to realize encryption control and key management from hardware solutions. Additional layers of the resource stack are provided as *virtual* abstractions, with the looming threat that the underlying layers may contain backdoors.

The use case concerns server-side encryption in a Cloud-storage system. The tenant or user ultimately *owns* the stored data. The tenant may also invoke encryption operations in the Cloud for its data. Thereby, the tenant relinquishes some control over the ownership of its data.

As the cryptographic operations take place in the Cloud platform, the keys become known to the platform. The provider may arrange this process in ways where it could gain much more access to the data than with client-side encryption. When the tenant supplies and manages the keys used for data encryption, the tenant retains a substantial level of control, as the Cloud platform could not on its own start decrypting data. On the other hand, if the tenant merely authenticates to the Cloud system and all keys are accessible to the Cloud platform, the provider can more easily access the data. The provider is not a homogeneous entity, there may be many different actors with potential access to the stored data. The main benefit of server-side encryption is to defend against provider-side actors with low-level access to stored data, who lack the privileges to access the encryption service. For example, this could be an employee with physical access to a disk rack in a data center but no access to servers.

2.1 Adapted Requirements Catalogue

For Use Case 1, Table 2.1 identifies the subset of requirements (from D1.1/D1.2) relevant for data ownership attributes. Table 2.1 first names and describes Use Case 1 requirements followed by their priority levels and the dependencies[1] across these requirements. Subsequently, the specific attribute of CIA/Performance/Functionality relevant to the requirement is identified. This table now forms the basis of identifying the threats for these requirements.

[1]Dependency relations between requirements are the direct relations between one or more requirements, where a requirement can depend on data or resources provided by another requirement.

Requirements Reference	Requirement Description	Priority	Dependencies	CIA	Performance	Functionality
REQ-UC1-IKM-1	CRUD operations for infrastructure keys	Medium		CA		✓
REQ-UC1-IKM-2	Policy-driven and automated infrastructure-key management	Medium	REQ-UC1-IKM-1	CA		✓
REQ-UC1-IKM-3	Support for standard APIs and protocols in infrastructure-key management	Medium	REQ-UC1-IKM-1			✓
REQ-UC1-IKM-4	Support for secure deletion of cryptographic material	Medium	REQ-UC1-IKM-2, REQ-UC1-IKM-3	C		✓
REQ-UC1-TKM-1	CRUD operations for tenant keys	High		CA		✓
REQ-UC1-TKM-2	Policy-driven and automated tenant-key management	Medium	REQ-UC1-TKM-1	CA		✓
REQ-UC1-TKM-3	Support for standard APIs and protocols in tenant-key management	High	REQ-UC1-TKM-1			✓
REQ-UC1-TKM-4	Support for secure deletion of cryptographic material	High	REQ-UC1-TKM-2, REQ-UC1-TKM-3	C		✓
REQ-UC1-SKM-1	Redundancy and fault-tolerance in key-management systems	High	REQ-UC1-TKM-1, REQ-UC1-IKM-1	A		✓
REQ-UC1-SKM-2	Scalable design of key-management system	Medium	REQ-UC1-TKM-1, REQ-UC1-IKM-1	CA	✓	

REQ-UC1-SKM-3	Key-management solutions support weakly consistent operations in cloud platform	Medium	REQ-UC1-TKM-1, REQ-UC1-IKM-1, REQ-UC1-SKM-1	CA		✓

Table 2.1: Relationship between Use Case 1 requirements and Confidentiality, Integrity, Availability, Performance and Functionality attributes with regards to Data Ownership.

2.2 Risk Assessment

Having characterized the data-ownership relevant requirements in Table 2.1, the requisite risk assessment is enumerated in Table 2.2. For each requirement, the first step is to identify the direct (principal) assumptions required for the requirement to hold. This is followed by the identification of the indirect (secondary) assumptions needed for the requirement to be supported. Then the violations that could occur between services and their expected (assumed) impact is specified. This is done by defining and measuring the type of risks causing these violations along with their violation likelihood and their severity.

Requirement Reference	Requirement Description	Direct Assumptions	Indirect Assumptions	Violation Likelihood (1/Low-10/High)	Assumed Impact	Threat Severity (1/Low-10/High)
REQ-UC1-IKM-1	CRUD operations for infrastructure keys	Trusted administrator	Key existence	7	Confidentiality violation for tenant data	10
REQ-UC1-IKM-1	CRUD operations for infrastructure keys	Secure storage for keys		3	Confidentiality violation for tenant data	10
REQ-UC1-IKM-1	CRUD operations for infrastructure keys	Key length sufficient		3	Confidentiality violation for tenant data	10
REQ-UC1-IKM-2	Policy-driven and automated infrastructure-key management	Policy language complete	Policy engine functional, automation present	4	Policies leak information about infrastructure keys, expose infrastructure keys	8
REQ-UC1-IKM-3	Support for standard APIs and protocols in infrastructure-key management	Correct protocol implementation		5	Denial of service, leakage of infrastructure information	10
REQ-UC1-IKM-4	Support for secure deletion of cryptographic material		Secure storage for keys	6	Keys exposed, data readable	10
REQ-UC1-TKM-1	CRUD operations for tenant keys	Trusted tenant administrator	Key existence, trusted infrastructure administrator	7	Confidentiality violation for tenant data	10
REQ-UC1-TKM-1	CRUD operations for tenant keys	Secure storage for keys		3	Confidentiality violation for tenant data	10

REQ-UC1-TKM-1	CRUD operations for tenant keys	Key length sufficient		3	Confidentiality violation for tenant data	10
REQ-UC1-TKM-2	Policy-driven and automated tenant-key management	Policy language complete	Policy engine functional, automation present	4	Policies leak information about tenant keys, expose tenant keys	8
REQ-UC1-TKM-3	Support for standard APIs and protocols in tenant-key management	Correct protocol implementation		5	Denial of service, leakage of tenant information	10
REQ-UC1-TKM-4	Support for secure deletion of cryptographic material		Secure storage for keys	6	Keys exposed, data readable	10
REQ-UC1-SKM-1	Redundancy and fault-tolerance in key-management systems	Redundancy and fault-tolerance		2	Denial of service	6
REQ-UC1-SKM-2	Scalable design of key-management system	System supports actual number of requests		2	Denial of service	6
REQ-UC1-SKM-3	Key-management solutions support weakly consistent operations in cloud platform	System correctly uses active keys		4	Older keys exposed	4
REQ-UC1-SKM-3	Key-management solutions support weakly consistent operations in cloud platform	System correctly uses active keys		4	Older keys used, data inaccessible	7

Table 2.2: Use Case 1 requirements and their direct and indirect assumptions as well as an estimated likelihood of direct assumptions being violated, assumed impact and threat severity.

3. Requirements Analysis from Use Case 2

Cloud computing requires data owners to outsource their data to a Cloud provider. To prevent unauthorized access, data has to be encrypted before outsourcing. However, the data still has to be utilizable for business purposes. Most business scenarios, like the ones involving complex supply chains, involve several parties which do not necessarily trust each other. This depicts a conflict: on one hand, sharing of data is a must in order to guarantee a better success of the business. On the other hand, substantial threats have to be taken care of in data sharing.

Therefore one goal of this Use Case is to come up with solutions allowing a data owner to share information without losing control over his data. This is realized by developing new systems for cooperation based on encrypted database technology. With such systems data owners should be empowered to outsource their data to the Cloud, while preserving security and functionality. Technology based on the search and aggregation of encrypted data ensures these functionalities. The technology also offers the data owner to selectively grant and revoke data access to their trusted business partners.

3.1 Adapted Requirements Catalogue

For Use Case 2, Table 3.1 identifies the subset of requirements (from D1.1/1.2) relevant for data ownership attributes. Table 3.1 first names and describes Use Case 2 requirements followed by their priority levels and the dependencies across these requirements. Subsequently, the specific attribute of CIA/Performance/Functionality relevant to the requirement is identified. This table now forms the basis of identifying the threats for these requirements.

Requirements Reference	**Requirement Description**	**Priority**	**Dependencies**	**CIA**	**Performance**	**Functionality**
REQ-UC2-AC1	Access Control per Client	High	REQ-UC2-KM1	C		✓
REQ-UC2-AC2	Access control per group of clients	High	REQ-UC2-KM2	C		✓
REQ-UC2-AC3	Access control per database cell	High	REQ-UC2-EQ2	C		✓
REQ-UC2-AC4	Access control matrix model	Medium		C		✓
REQ-UC2-AC5	Access grant and revoke by administrator	Low	REQ-UC2-EQ3	CI		✓
REQ-UC2-AC6	Access control enforced by client	High	REQ-UC2-KM3	CI		✓
REQ-UC2-KM1	One key per client	High		CI		✓
REQ-UC2-KM2	Group key management	High		CI		✓
REQ-UC2-KM3	Client key securely stored at client only	High		CI		✓
REQ-UC2-KM4	Group keys derivable	High	REQ-UC2-KM2	CI		✓
REQ-UC2-EQ1	Encryption schemes	High		C	✓	
REQ-UC2-EQ2	Adjustable onion encryption	High	REQ-UC2-EQ1	C	✓	
REQ-UC2-EQ3	Proxy re-encryption, Query rewriting, Post-processing	High		C	✓	
REQ-UC2-EQ4	Support for different keys	High	REQ-UC2-EQ3	C	✓	

Table 3.1: Relationship between Use Case 2 requirements and Confidentiality, Integrity, Availability, Performance and Functionality attributes with regards to Data Ownership.

3.2 Risk Assessment

Having characterized the data-ownership relevant requirements in Table 3.1, the requisite risk assessment is enumerated in Table 3.2. For each requirement, the first step is to identify the direct (principal) assumptions required for the requirement to hold. This is followed by identification of the indirect (secondary) assumptions needed for the requirement to be supported. Then the violations that could occur between services and their expected (assumed) impact is specified. This is done by defining and measuring the type of risks causing these violations along with their violation likelihood and their severity.

Requirement Reference	Requirement Description	Direct Assumptions	Indirect Assumptions	Violation Likelihood (1/Low-10/High)	Assumed Impact	Threat Severity (1/Low-10/High)
REQ-UC2-AC1	Access Control per Client	User Authentication		5	Basic assumption violated	8
REQ-UC2-AC2	Access control per group of clients	User Groups exist		3	User inconvenience	1
REQ-UC2-AC3	Access control per database cell	Cells are controllable		3	Exposure of larger structures, e.g. tables or columns, potentially resulting in partial confidentiality violation	4
REQ-UC2-AC4	Access control matrix model	Independent of time and workflow		4	User inconvenience	1
REQ-UC2-AC5	Access grant and revoke by administrator	Administrator for group maintenance available		3	User inconvenience	1
REQ-UC2-AC6	Access control enforced by client	Trusted Client		5	Confidentiality violation for user data	5
REQ-UC2-KM1	One key per client	Key length sufficient, keys are renewable	Secure Storage for Key	2	Keys guessable, global confidentiality violation	10
REQ-UC2-KM2	Group key management	Group can secure key, Key is renewable, key length is sufficient		8	Confidentiality violation for group data	5

REQ-UC2-KM3	Client key securely stored at client only	Secure Storage		2	Confidentiality violation for user data	5
REQ-UC2-KM4	Group keys derivable	Cryptographic assumptions hold		3	Disaster, global confidentiality violation	10
REQ-UC2-EQ1	Encryption schemes	Cryptographic assumptions hold	Key is secret (i.e. KM1)	3	Disaster, global confidentiality violation	10
REQ-UC2-EQ2	Adjustable onion encryption	Set of supported queries is sufficient, Scalability is given		4	Performance problems	2
REQ-UC2-EQ3	Proxy re-encryption, Query rewriting, Post-processing	Security enforcement is possible for Multi user environment		3	User has access to data which was supposed to be revoked, resulting in partial confidentiality violation	4
REQ-UC2-EQ4	Support for different keys	Set of supported queries is sufficient for Multi user, Scalability is given for Multi user		5	Performance problems	2

Table 3.2: Use Case 2 requirements and their direct and indirect assumptions as well as an estimated likelihood of direct assumptions being violated, assumed impact and threat severity.

4. Requirements Analysis from Use Case 3

This Use Case considers the application of data protection in multi-cloud environments including federated secure cloud storage. It will offer data protection as a service via a multi-tenant cloud service store that enables customers to protect their data stored on multiple cloud platforms. As such this Use Case is particularly applicable for Managed Security Service Providers and for Cloud Service Provider customers who want to control the protection of data at rest across multiple cloud environments and apply uniform data protection and data access policies for heterogeneous data stored across internal, private and public clouds.

The data is protected by leveraging a cloud-based data protection service for encrypting independently of the cloud provider hosting it, and ensuring that cloud service providers have no access to the encryption keys or its protection and access control policies. The encrypted data can be stored on multiple cloud storage services like block storage, object stores and Big Data clusters.

4.1 Adapted Requirements Catalogue

For Use Case 3, Table 4.1 identifies the subset of requirements (from D1.1/D1.2) relevant for data ownership attributes. Table 4.1 first names and describes Use Case 3 requirements followed by their priority levels and the dependencies across these requirements. Subsequently, the specific attribute of CIA/Performance/Functionality relevant to the requirement is identified. This table now forms the basis of identifying the threats for these requirements.

Requirements Reference	Requirement Description	Priority	Dependencies	CIA	Performance	Functionality
REQ-UC3-KM-1	Each tenant should be provisioned with an instance of a key management service from the cloud service store	High	REQ-UC3-AC-7, REQ-UC3-SO-1, REQ-UC3-SO-2	CIA		✓
REQ-UC3-KM-2	The tenants should be able to generate, insert, retrieve and remove keys from their key management service	Medium	REQ-UC3-KM-1	CI		✓
REQ-UC3-KM-3	The key management service should be able to offer different key types and generation algorithms to each tenant, e.g., AES128, AES256, 3DES etc.	Low	REQ-UC3-KM-1, REQ-UC3-DE-1, REQ-UC3-DE-3	IA		✓
REQ-UC3-KM-4	Only the tenants should be able to create and manage the keys	High	REQ-UC3-KM-1	CI		✓
REQ-UC3-KM-5	The cloud service providers should have no access or visibility of the tenants' keys	High	REQ-UC3-KM-1	CI		✓
REQ-UC3-KM-6	The tenants should be able to cache their keys on trusted virtual machines or gateways in order to outsource or improve performance of the encryption and decryption process	Low	REQ-UC3-KM-1, REQ-UC3-DE-2	A		✓
REQ-UC3-AC-1	Each tenant should be provisioned with an instance of an access control service from the cloud service store	High	REQ-UC3-SO-1, REQ-UC3-SO-6	CA		✓

REQ-UC3-AC-2	The tenants should be able to create, delete and modify access control policies from their instance of the access control service	Medium	REQ-UC3-AC-1	CA		✓
REQ-UC3-AC-3	The access control service should be able to offer use of different system and data attributes for the construction of a security rule, e.g., file-system, user, application, and time attributes.	Low	REQ-UC3-AC-1	IA		✓
REQ-UC3-AC-4	Only the tenants should be able to create and manage their access control policies	High	REQ-UC3-AC-1	CA		✓
REQ-UC3-AC-5	The cloud service providers should have no access or visibility of the tenants' access control policies	High	REQ-UC3-AC-1	CI		✓
REQ-UC3-AC-6	All data protection operations should be governed by access control policies by either approving or denying access to the required keys	High	REQ-UC3-AC-1, REQ-UC3-KM-1	CIA		✓
REQ-UC3-AC-7	The access control service of tenants should be tightly coupled with their key management service, such that no key can be utilised without an approving access control policy	High	REQ-UC3-AC-1, REQ-UC3-KM-1	IA		✓
REQ-UC3-SO-1	Each tenant should be provisioned with a cloud service store account	High		A		✓

REQ-UC3-SO-2	The service store should provide the tenants with access to the storage services of multiple cloud service providers	Medium		A		✓
REQ-UC3-SO-3	The service store should be able to offer block storage service to the tenants	High		A		✓
REQ-UC3-SO-4	The service store should be able to offer object storage service to the tenants	High		A		✓
REQ-UC3-SO-5	The service store should be able to offer Big Data storage service (HDFS) to the tenants	High		A		✓
REQ-UC3-SO-6	The tenants should be able to enable or disable the use of data protection service on the storage service of their choice	Medium		CA		✓
REQ-UC3-SO-7	The service store should be able to offer key management as a service to the tenants	High		CA		✓
REQ-UC3-SO-8	The service store should be able to offer access control as a service to the tenants	High		CA		✓
REQ-UC3-DE-1	The core encryption process should only be controlled and managed by the tenant	High		CA		✓

REQ-UC3-DE-2	The tenant should be able to deploy and manage the core encryption process on trusted virtual machines or gateways as an agent or plug-in	Medium		CA		✓
REQ-UC3-DE-3	The core encryption process should be FIPS 140 compliant	Medium		CI	✓	
REQ-UC3-DE-4	The encryption agent or plug-in should be able to access the tenant's key management service and access control service	Medium		CI		✓
REQ-UC3-DE-5	The keys should only be released to the encryption agent or plug-in upon approval of an access control policy	High		CA		✓

Table 4.1: Relationship between Use Case 3 requirements and Confidentiality, Integrity, Availability, Performance and Functionality attributes with regards to Data Ownership.

4.2 Risk Assessment

Having characterized the data-ownership relevant requirements in Table 4.1, the requisite risk assessment is enumerated in Table 4.2. For each requirement, the first step is to identify the direct (principal) assumptions required for the requirement to hold. This is followed by identification of the indirect (secondary) assumptions needed for the requirement to be supported. Then the violations that could occur between services and their expected (assumed) impact is specified. This is done by defining and measuring the type of risks causing these violations along with their violation likelihood and their severity.

Requirement Reference	Requirement Description	Direct Assumptions	Indirect Assumptions	Violation Likelihood (1/Low-10/High)	Assumed Impact	Threat Severity (1/Low-10/High)
REQ-UC3-KM-1	Each tenant should be provisioned with an instance of a key management service from the cloud service store	Tenant has an account on the cloud service store	Tenant has the ability to subscribe to the DPaaS	1	Denial of Service	2
REQ-UC3-KM-2	The tenants should be able to generate, insert, retrieve and remove keys from their key management service	Tenant has access to the KMS	KMS is operational	2	Denial of Service	2
REQ-UC3-KM-3	The key management service should be able to offer different key types and generation algorithms to each tenant, e.g., AES128, AES256, 3DES etc.	KMS is operational and available		1	Denial of Service	2
REQ-UC3-KM-4	Only the tenants should be able to create and manage the keys	Multi-tenant feature of the KMS is operational and available		2	Confidentiality and Integrity violation of keys	10

REQ-UC3-KM-5	The cloud service providers should have no access or visibility of the tenants' keys	KMS is hosted in a trusted environment	KMS is in a hardened OS or Sandbox	1	Confidentiality and Integrity violation of keys	10
REQ-UC3-KM-6	The tenants should be able to cache their keys on trusted virtual machines or gateways in order to outsource or improve performance of the encryption and decryption process	KMS can generate cacheable keys, unique to particular hosts		2	Degradation of encryption and decryption performance	1
REQ-UC3-AC-1	Each tenant should be provisioned with an instance of an access control service from the cloud service store	Tenant has an account on the cloud service store	Tenant has the ability to subscribe to the DPaaS	2	Denial of Service	2
REQ-UC3-AC-2	The tenants should be able to create, delete and modify access control policies from their instance of the access control service	Tenant has access to the AC service	AC service is operational	2	Denial of Service	2

REQ-UC3-AC-3	The access control service should be able to offer use of different system and data attributes for the construction of a security rule, e.g., file-system, user, application, and time attributes.	AC service is operational and available		1	Denial of Service	2
REQ-UC3-AC-4	Only the tenants should be able to create and manage their access control policies	Multi-tenant feature of the AC service is operational and available		2	Confidentiality and Integrity violation of keys	10
REQ-UC3-AC-5	The cloud service providers should have no access or visibility of the tenants' access control policies	AC service is hosted in a trusted environment	AC service is in a hardened OS or Sandbox	2	Confidentiality and Integrity violation of keys	10
REQ-UC3-AC-6	All data protection operations should be governed by access control policies by either approving or denying access to the required keys	Key release is tightly coupled with AC service		5	Partial loss of data access - Denial of Service	8

REQ-UC3-AC-7	The access control service of tenants should be tightly coupled with their key management service, such that no key can be utilised without an approving access control policy	Key release is not possible without correct policy		5	Partial loss of data access - Denial of Service	8
REQ-UC3-SO-1	Each tenant should be provisioned with a cloud service store account	Each tenant is given a unique identifier		2	Denial of Service	2
REQ-UC3-SO-2	The service store should provide the tenants with access to the storage services of multiple cloud service providers	Service store has access profiles of relevant cloud service provider		3	Denial of Service, Partial loss of data access	3
REQ-UC3-SO-3	The service store should be able to offer block storage service to the tenants	Service store is able to provision block storage from relevant cloud service provider		3	Denial of Service, Partial loss of data access	3
REQ-UC3-SO-4	The service store should be able to offer object storage service to the tenants	Service store is able to provision object storage buckets from relevant cloud service provider		3	Denial of Service, Partial loss of data access	3

REQ-UC3-SO-5	The service store should be able to offer Big Data storage service (HDFS) to the tenants	Service store is able to provision HDFS service from relevant cloud service provider		3	Denial of Service, Partial loss of data access	3
REQ-UC3-SO-6	The tenants should be able to enable or disable the use of data protection service on the storage service of their choice	Service store can subscribe or unsubscribe from the DPaaS		1	Denial of Service	5
REQ-UC3-SO-7	The service store should be able to offer key management as a service to the tenants	KMS has an operational plug-in for the Service Store		4	Denial of Service	10
REQ-UC3-SO-8	The service store should be able to offer access control as a service to the tenants	AC Service has an operational plug-in for the Service Store		4	Denial of Service	10
REQ-UC3-DE-1	The core encryption process should only be controlled and managed by the tenant	Only the tenant can specify the target storage services where data is stored		4	Confidentiality and Integrity violation of tenant data	10
REQ-UC3-DE-2	The tenant should be able to deploy and manage the core encryption process on trusted virtual machines or gateways as an agent or plug-in	Service store has configuration management capability for VMs and gateways	An agent or plug-in for target VMs or gateways	4	Denial of Service	5

REQ-UC3-DE-3	The core encryption process should be FIPS 140 compliant	DPaaS components are compliant to industry standards		3	Possible confidentiality and integrity violation due to weak algorithms	7
REQ-UC3-DE-4	The encryption agent or plug-in should be able to access the tenant's key management service and access control service	These services are accessible over an encrypted channel using protocols like SSL/TLS		4	Loss of confidentiality, integrity and availability	10
REQ-UC3-DE-5	The keys should only be released to the encryption agent or plug-in upon approval of an access control policy	Key release is not possible without correct policy		5	Partial loss of data access - Denial of Service	8

Table 4.2: Use Case 3 requirements and their direct and indirect assumptions as well as an estimated likelihood of direct assumptions being violated, assumed impact and threat severity.

The analysis presented in the table is necessarily subjective but the critical nature of the data protection offered, plus the fact that policy control, access and encryption are offered at the enterprise level, means that the impact of a successful attack is likely to be severe for customers. Hence, there is a huge incentive for providers of these services to ensure that they are protected to the highest degree, and that customers are aware that they should only procure such services from trusted partners with appropriate experience, compliance and deep cyber security expertise. This analysis has helped to identify those requirements from the Use Case that are of highest risk and this will inform the implementation techniques to ensure maximal protection of these features.

5. Requirements Analysis from Use Case 4

The Use Case 4 considers the use of an elastic Cloud Service Provider (CSP). As such this Use Case is of particular relevance for Cloud service brokers or intermediaries offering a secure Cloud data storage capability to their customers while possibly leveraging other Cloud providers for storing this data and ensuring that data is protected from such other Cloud providers and other users.

Therefore, this Use Case 4 has to provide data protection in Clouds and multi-Cloud environment that enables users to protect their data without the intervention of the Cloud Provider. This situation allows users to have the control of their data while providing flexibility to the services provided by Cloud Service Providers. Moreover, this Use Case empowers the use of other Cloud Providers (Third Parties), so that a Cloud Provider could use any other Third Party's Cloud if demand expands. Using ESCUDO-CLOUD Middleware the data is protected by encryption at client side. This is the case where the data owner is the one who knows and manages the data security while the Cloud Service Provider has no access to the encryption. Once the user is logged in the ESCUDO-CLOUD Middleware, this encryption is performed through the browser allowing users to access to his data in a transparent way, without the necessity of installing additional software.

5.1 Adapted Requirements Catalogue

For Use Case 4, Table 5.1 identifies the subset of requirements (from D1.1/D1.2) relevant for data ownership attributes.Table 5.1 first names and describes Use Case 4 requirements followed by their priority levels and the dependencies across these requirements. Subsequently, the specific attribute of CIA/Performance/Functionality relevant to the requirement is identified. This table now forms the basis of identifying the threats for these requirements.

Requirements Reference	Requirement Description	Priority	Dependencies	CIA	Performance	Functionality
REQ-UC4-AC1	Each user should have a unique username and password to enter in the system through a web portal. The user will login in the middleware using his credentials, and only with them will be able to access his/her stored data files.	High		C		✓
REQ-UC4-AC2	By using the agent, it should be possible to remember the credentials. When a user accesses for the first time the agent installed in his device (computer, mobile...) he will login with his username and password, but he should not need to introduce them again.	High		C		✓
REQ-UC4-AC3	Each user will have a unique username/password to login to the middleware. The middleware will control what data the user can access and what operations the user can perform on that data. For example, a user with read privileges should not be able to modify a data file.	High		C		✓
REQ-UC4-AC4	Only the file owner should be able to authorize another user for reading or modifying his data files, that is to say that the file owner will assign the role of the rest of the users.	High		A		✓

REQ-UC4-AC5	Only the administrator of the system (the ESCUDO-CLOUD middleware) should be able to enable the access of a locked user.	High	REQ-UC4-AC6	A		✓
REQ-UC4-AC6	Each user should have a limit of attempts to access the system. For example, if a user introduces five wrong credentials, he will be locked.	High	REQ-UC4-AC5	A		✓
REQ-UC4-SS1	The Cloud (or storage service) should have capacity enough to store the data files of the users.	High	REQ-UC4-SS3	A	✓	
REQ-UC4-SS2	The storage services should be accessible for the end users through ESCUDO-CLOUD middleware.	High		A	✓	✓
REQ-UC4-SS3	By using an elastic Cloud should be possible to adapt the capacity of the Cloud to the user requirements.	High	REQ-UC4-SS1	A	✓	

REQ-UC4-SS4	The storage service should comply with ”EU Directive 95/46/EC - The Data Protection Directive” The personal data is defined under Article 2 a: *”personal data shall mean any information relating to an identified or identifiable natural person (”data subject”); an identifiable person is one who can be identified, directly or indirectly, in particular by reference to an identification number or to one or more factors specific to his physical, physiological, mental, economic, cultural or social identity”.*	High	REQ-UC4-SS5	CI	✓	
REQ-UC4-SS5	The CSP should implement appropriate technical and organizational measures to protect personal data against accidental or unlawful destruction or accidental loss, alteration, unauthorized disclosure or access, in particular where the processing involves the transmission of data over a network, and against all other unlawful forms of processing.	High	REQ-UC4-SS4, REQ-UC4-DE1, REQ-UC4-DE2, REQ-UC4-DE3, REQ-UC4-DE4	CIA	✓	

REQ-UC4-DE1	The files should be stored encrypted. Only the user with the correct privileges will be able to decrypt the information, not even the administrator of the server will be able to access the content of the user data files.	High	REQ-UC4-DE2, REQ-UC4-DE3, REQ-UC4-SS5	C	✓	
REQ-UC4-DE2	The data files should remain encrypted on the CSP until the user accesses them through a web browser.	High	REQ-UC4-DE1, REQ-UC4-SS5	C	✓	
REQ-UC4-DE3	The data files should remain encrypted on the CSP until the user synchronizes the server with the agent installed on his device.	High	REQ-UC4-DE1, REQ-UC4-SS5	C	✓	
REQ-UC4-DE4	An end user should be able to securely download a data file from the CSP through the web browser. To fulfil this purpose, the client should have enough capacity to make an ”on flight decryption”.	High	REQ-UC4-SS5	CIA	✓	✓

Table 5.1: Relationship between Use Case 4 requirements and Confidentiality, Integrity, Availability, Performance and Functionality attributes with regards to Data Ownership.

5.2 Risk Assessment

Having characterized the data-ownership relevant requirements in Table 5.1, the requisite risk assessment is enumerated in Table 5.2. For each requirement, the first step is to identify the direct (principal) assumptions required for the requirement to hold. This is followed by identification of the indirect (secondary) assumptions needed for the requirement to be supported. Then the violations that could occur between services and their expected (assumed) impact is specified. This is done by defining and measuring the type of risks causing these violations along with their violation likelihood and their severity.

Requirement Reference	Requirement Description	Direct Assumptions	Indirect Assumptions	Violation Likelihood (1/Low-10/High)	Assumed Impact	Threat Severity (1/Low-10/High)
REQ-UC4-AC1	Access Control to the web portal	Proper user authentication and proper rights for the logged user	Corporate password policy enforcement	5	Data confidentiality is violated	8
REQ-UC4-AC2	Save credentials in the device	Credentials accessible only by the owner		3	Data confidentiality is violated	8
REQ-UC4-AC3	Access control to middleware	Proper user authentication and proper rights for the logged user	Corporate password policy enforcement	5	Data confidentiality is violated	8
REQ-UC4-AC4	Access to shared files only with permission of the file owner	A file owner exists		3	Shared file confidentiality is violated	4
REQ-UC4-AC5	Access grant by administrator for locked users	Administrator is the only entity that can assign permission	Malicious user was previously blocked	3	User has access to data that was supposed to be revoked, resulting in partial confidentiality violation	7
REQ-UC4-AC6	Limit of failed attempts	There is a maximum number of attempts	Key is sufficiently safe	2	User data confidentiality is violated	8
REQ-UC4-SS1	Ensure Cloud capacity	Check available capacity in the Cloud before store user data		3	Failure to upload user data	6

REQ-UC4-SS2	Storage access control through middleware	Users cannot access Cloud storage bypassing the middleware		3	Total access to storage systems	10
REQ-UC4-SS3	Eastic Cloud capacity	Capacity can be adapted to meet user requeriments	There is sufficient capacity in the Cloud	2	User data cannot be stored in the Cloud	6
REQ-UC4-SS4	Comply data protection directive (EU 95/46/EC)	There are enough mechanisms to protect user personal data		2	Disclosure of personal data	7
REQ-UC4-SS5	Data recovery control	A backup mechanism exists	The backup system is resilient	2	Loss of user data	6
REQ-UC4-DE1	Only data owner can decrypt data	Data are encrypted in the Cloud and user possesses the encryption key		3	User data confidentiality violation	9
REQ-UC4-DE2	Store data encrypted in the Cloud	Client side encryption		1	User data confidentiality violation	6
REQ-UC4-DE3	Server synchronization before decryption	Prior stage to user-data interaction		1	User cannot access its data	3
REQ-UC4-DE4	Secured file download through the web browser	Client side decryption		1	Data confidentiality, integrity and availability violation	8

Table 5.2: Use Case 4 requirements and their direct and indirect assumptions as well as an estimated likelihood of direct assumptions being violated, assumed impact and threat severity.

6. Dependency Analysis Across Requirements

In the preceding chapters, the Use Case requirements and risk assessment tables of each Use Case (individually and collectively) were specified to establish the viability of a requirements based threat analysis. However, this process was manual and also provides no assurance on completeness. Consequently, the intent in this section is to develop techniques to systematically conduct the process of ascertaining and visualizing the dependencies across the Use Case requirements to identify threats. Based on the tables from Chapter 2-6, the Use Case requirements are modelled using a graphical mapping in a hierarchical structure, as shown in Figure 6.1. The root of the structure defines the main container for the requirements (specified as $level_0$ in Figure 6.1). The intermediate level ($level_1$) represents the main services which are the primary link to the security framework used by the Cloud provider and the customer. The lowest level ($level_2$) in the structure represents the actual requirements committed by the Cloud provider and offered to the Cloud customer. The structure shows the existence of dependency relationships across different Use Case requirements as depicted in Figure 6.1.

These dependencies increase the difficulty for the customers when specifying their requirements, as these relations can easily introduce conflicts. For instance, a customer may require an unachievable level[1] of a dependent service which makes customer's requirements unsatisfiable. It also becomes hard to manually detect and identify the causes of these conflicts as one customer requirement may influence or be influenced by other requirements. Furthermore, a provider can agree on providing an unachievable level of dependent service according to the customer requirement. However, the provider will not be able to fulfill this requirement which results in a service violation.

In addition, even after these requirements are committed by the Cloud provider and offered to the Cloud customer as services, a security breach causing a violation of one of these services can lead to the violation of other services (cf., Section 6.4). Depending on the nature of the incident, these violations can be anything from low-risk to highly critical. Consequently, an explicit knowledge about dependencies between requirements and their threat severity is needed to support the management of Cloud services by both entities (the Cloud provider and/or the customer).

This chapter aims to solve the aforementioned issues by presenting a framework for the analysis of the dependency relationships across different Use Case requirements with the handling of all conflicts. This is done by:

1) Presenting a dependency representation model for validating each Use Case requirement by checking the existence of conflicts that occur due to different dependency relations between requirements. The process of analyzing conflicts is both iterative and interactive.

[1] Unachievable service level is the service level that requires a certain resource which is not provided by the corresponding dependent service.

2) Assisting customers who could not resolve conflicts and thus could not specify their requirements by structuring the requirements using Design Structure Matrix (DSM)[2]. This makes the dependencies across each Use Case requirements explicit and traceable regardless of the number of requirements.

3) Allowing providers to identify the most critical services that if violated, the system is more susceptible to security breaches. This is achieved using the DSM dependent ordered structure to represent the services from the least to the most dependent (i.e., Highly critical to low-risk).

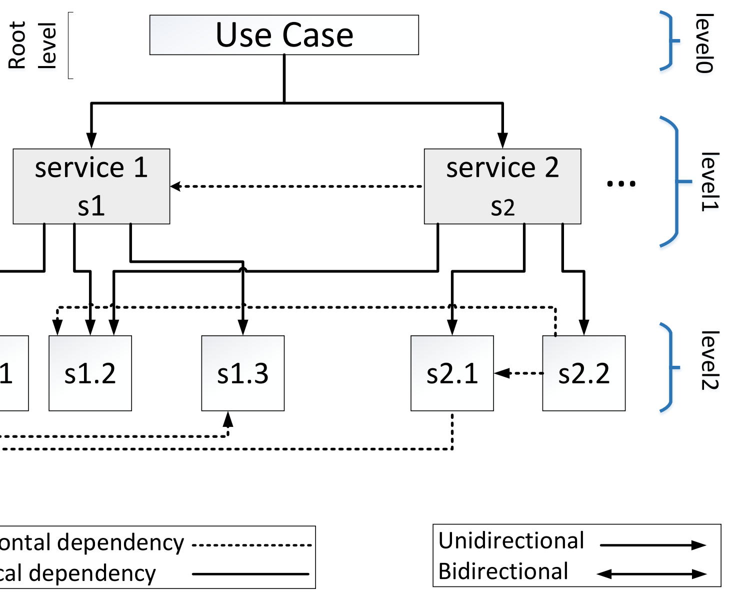


Figure 6.1: A Use Case requirements arranged in a hierarchy structure showing the dependencies between them

6.1 Background: Concepts, State of the Art and Innovation

6.1.1 Service Dependencies

A service dependency is a directed relation between the services offered in Cloud scenarios. It is expressed as a $1 : n$ relationship where one service (termed as dependent) depends on one or multiple services (termed as antecedent). A service can depend on data or resources provided by another service. A service s_1 is dependent on service s_2 if the provisioning of s_1 is conditional to the provisioning of s_2. Explicit knowledge about dependencies is needed to support the management of services by both CSPs and customers. Several types of dependencies are used in literature such as Quality of Service (QoS), price, resource and time dependencies[3] [WSS10]. We consider resource dependencies which are validated by matching the security values of the dependent and antecedent services.

[2]Structuring the requirements in a precise order according to their level of dependency.

[3]In project management, a resource dependency is described by a list of resources regarding which the dependency exists and time dependencies are described using time operators [SWSS09]

We classify dependencies based on their occurrence between services and requirements at the same hierarchical level (horizontal dependencies), as well as between different levels in the hierarchical structure (vertical dependencies) [WS09] as shown in Figure 6.1. Dependencies can be further classified into direct and indirect dependencies. Indirect dependencies occur between services which do not directly interact with each other, but where a transitive relationship exists via an intermediate service. Figure 6.1 illustrates different dependencies such as, service s_2 depends on service s_1 in the same hierarchical level, which represents a direct horizontal dependency. Furthermore, s_1 depends on $s_{1.1}$, $s_{1.2}$, and $s_{1.3}$, which specifies direct vertical dependencies. As a result, s_2 depends on $s_{1.1}$, $s_{1.2}$, and $s_{1.3}$ indirectly through s_1 (i.e., Transitivity property). In many cases horizontal and vertical dependencies occur at the same time and both dependencies affect the whole composition hierarchy.

All the dependencies explained so far are considered unidirectional dependencies. Other dependencies such as bidirectional (interdependent relations between services) may occur as well. Bidirectional dependency occurs between services $s_{1.1}$ and $s_{2.1}$ if the provisioning of $s_{1.1}$ is conditional to the provisioning of $s_{2.1}$ and at the same time the provisioning of $s_{2.1}$ is conditional to the provisioning of $s_{1.1}$. Dependencies can be further classified into direct and indirect dependencies.

6.1.2 State of the Art and ESCUDO-CLOUD Innovation

At the state of the art, threat modeling is a broad and extensively studied area. The primary approach therein is Microsoft's STRIDE (Spoofing, Tampering, Repudiation, Information-Disclosure, DoS and Elevation-of-Privilege) that models and explores an attack surface from an entry point to the attack realization. The multiple alternate techniques vary on STRIDE (e.g., PASTA: Process for Attack Simulation and Threat Analysis, VAST: Visual, Agile Simple Threat Modeling, etc.) depending on the abstraction level of the attack surface which can be used as requirements at the software component level, as system ports, at middleware interfaces or the protocol stack. The overall intent behind these vulnerability analyses is to create an attack tree in which the propagation of an attack can be studied. An extensive discourse on such threat modeling appears at the OWASP website (`www.owasp.org/index.php/Threat_Risk_Modeling`) and also at SANS (`www.sans.org`). A number of EC projects have also applied such techniques at varied system levels e.g., ABC4TRUST (`www.abc4trust.eu`) applied a STRIDE variant for conducting vulnerability analysis at the component interface levels. SPECS (`www.specs-project.edu`) applied a combination of STRIDE and PASTA at the API level. However, there is no existing approach that can utilize the functional requirements of Cloud services as a basis for threat analysis as conducted in T2.4. Also a significant drawback of all existing threat modeling approaches is their lack of considering implicit of explicit dependencies between services. In addition, most approaches also entail a high degree of manual modeling of the attack tree. Thus, our presented contribution (also published in [TMT+16]) represents a significant innovation enhancement to the state of the art and state of the practice.

6.2 Dependency Management Approach

After the mapping of the Use Case requirements in a hierarchical structure (depicted in Figure 6.1), this structure is used to develop the dependency management approach presented in this chapter. As an overview of the presented approach, the dependency management is performed in progressive stages, as shown in Figure 6.2.

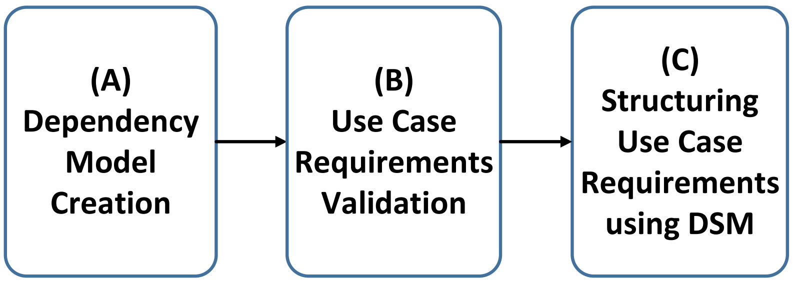

(A) Dependency Model Creation
(B) Use Case Requirements Validation
(C) Structuring Use Case Requirements using DSM

Figure 6.2: Dependency management approach stages

A dependency model is created in Stage (A) to capture information about the Use Case requirements (i.e., Services) and the subsequent dependencies. Subsequently in Stage (B), this model is specified using a machine readable format to allow automated validation for checking service conflicts and different services compatibility issues. Finally in Stage (C), the validated services are structured using the Design Structure Matrix (DSM) (also known as "Dependency Structure Matrix") [Ste81] depicting dependencies between services as an ordered list.

6.2.1 Stage (A). Dependency Model Creation

The approach for managing service dependencies builds on a dependency model, which is used to capture information about the dependencies across services. In order to model service dependencies, it is important to first derive the expected requirements that the dependency model should support:

1. *Different dependency types*. The dependency model should support different types of dependencies as well as various dependencies classifications (e.g., horizontal, vertical, unidirectional and bidirectional dependencies).

2. *Multiple dependencies*. One service can depend on several other services. These dependencies could be same or of different types.

3. *Dependency model validation*. It should be possible to automatically validate the dependency model to avoid inconsistencies and conflicts.

Handling all the dependencies between services is a very time consuming and complex task. Therefore, a dependency model is created for the Use Case requirements hierarchical structure (cf., Figure 6.1) to cover all identified dependencies within this structure. This model is used to capture information about services and the dependencies that occur between them. We model each Use Case hierarchical structure by a tuple $(S, l, \rightarrow_S, v)$ where:

- S is a set of services s with associated hierarchy levels $l(s) \in \{0, 1, \ldots, n\}$. Each Use Case requirements structure is composed of three hierarchical levels (i.e., $n = 2$) as shown in Figure 6.1 (i.e., $level_0$, $level_1$ and $level_2$).

- Each service s consists of m different metric values v. Each value implies a different security level offered by the provider and required by the customer.

- $v : S \mapsto V$ is an assignment of values in V to services committed by the Cloud provider and offered to the Cloud customer, where V is the set of all metric values of each service in S with $l(s) = 2$.

- $\rightarrow_S \subseteq S \times (S)$ models service dependencies.

- We write $s_1 \rightarrow_S s_2$ if s_1 (dependent service) depends on s_2 (antecedent service).

- Constraints on the services dependency relation are specified using a constraint set $C_v^{\rightarrow_S} \subseteq S \times S \times \{=, \neq, <, \leq, >, \geq\}$. A constraint $(s_1, s_2, \equiv) \in C_v^{\rightarrow_S}$ is satisfied if the values of s_1 and s_2 are related by the given comparison, i.e., $v(s_1) \equiv v(s_2)$. A dependency relation $s_1 \rightarrow_S s_2$ is called valid, if the relation satisfies all its constraints, i.e., $\forall (s_1', s_2', \equiv) \in C_v^{\rightarrow_S}.(s_1 = s_1'$ and $s_2 = s_2') \Rightarrow v(s_1) \equiv v(s_2)$.

- We write the transitive closure of $\rightarrow_S$ as $\rightarrow_S^+$, i.e., $s_1 \rightarrow_S^+ s_2$ if $s_1 \rightarrow_S s_2$ or $\exists s_3 \in S.s_1 \rightarrow_S s_3$ and $s_3 \rightarrow_S^+ s_2$. A dependency $s_1 \rightarrow s_2$, between two services $s_1, s_2 \in S$ is called symmetric if also $s_2 \rightarrow s_1$. Otherwise, $s_1 \rightarrow s_2$ is called non-symmetric. In other words, s_1 and s_2 are symmetrically dependent if $s_1 \rightarrow s_2$ and $s_2 \rightarrow s_1$.
  Note that it is possible that $s_1 \rightarrow s_2$ is symmetric while the relation $\rightarrow$ is non-symmetric. We explain this using an example, let us consider $\rightarrow = \{(s_1, s_2, w), (s_2, s_1, w), (s_1, s_3, w)\}$ which is a non-symmetric relation and where we have that $s_1 \rightarrow s_2$ is symmetric.

- Each Use Case hierarchical structure has to satisfy the following constraints:

  i) Services do only depend on services of the same or the next lower hierarchy level: $\forall s_1, s_2 \in S.s_1 \rightarrow_S s_2 \Rightarrow l(s_1) = l(s_2)$ or $l(s_1) + 1 = l(s_2)$

  ii) Services do not depend on themselves: $\forall s \in S.\neg s \rightarrow_S s$

6.2.2 Stage (B). Validation

A meta-model is developed based on the dependency definitions. This meta-model allows the description of services along with the information on the Use Case requirements drafted for it. The meta-model is specified using a machine readable format (allowing fully automatic validation) such as an XML data structure using an XML Schema. In this Schema, dependency relations between services are modeled by including the involved services and their roles as dependent or antecedent as well as their values. Moreover, the constraint comparison is extracted and modeled in this Schema.

The extract of a Use Case dependency model with two services (named "s_1" (s_{IKM-1}) and "s_2" (s_{IKM-2})) and the dependency relation between them is shown in Listing 1. In the Listing, $s_{IKM-2} \rightarrow_S s_{IKM-1}$ and the two services levels are modeled as $v(s_{IKM-2})$ and $v(s_{IKM-1})$, respectively. The requirement is that the security level of s_{IKM-2} is equal to the security level of s_{IKM-1}, i.e., $v(s_{IKM-2}) = v(s_{IKM-1})$. This requirement is modeled as $(s_{IKM-1}, s_{IKM-2}, =) \in C_v^{\rightarrow_S}$.
Note that all service levels (e.g., $level_2$, $level_3$, ...) are modeled as numerical values. These numerical values are the services values in the XML Schema shown in Listing 1.

```
<Use−Case UCid=‘‘UC1">
<dependencyModel Depmodelid=‘‘depmod_1">
    <serviceDependency Depid=‘‘dep−1" type=‘‘unidirectional">
        <dependent depserviceid=‘‘s2_UC1" value =‘‘3"/>
        <antecedent antserviceid=‘‘s1_UC1" value=‘‘3"/>
        <constraint> eq </constraint>
    </serviceDependency>
</dependencyModel>
</Use−Case>
```

Listing 6.1: Excerpt of dependency model of a UC1 requirements

Following the development of the dependency model, the services are validated as depicted in Figure 6.3. The validation is done by first extracting the Use Case ID, the dependency model ID and the dependency ID of each two dependent services (each dependency relation in the same dependency model has a unique ID) defined in the XML Schema. Furthermore, for each dependent relation the antecedent and dependent service's values are extracted. This entails extracting the constraint comparative (i.e., $=, \neq, \leq, \geq$) and checking if the two dependent service values satisfy the constraint. If the constraint between dependent services is not satisfied, the validation scheme shows a conflict between these two services. The dependency ID and the dependent service ID of the affected services are saved in a list, while the evaluation is continued to determine further conflicts. At the end of this stage a list of all conflicts found in the provider offered services with the conflicts explanation is sent to the provider in order to resolve these conflicts and resubmit his/her services again to be validated. Similarly, a list of all conflicts found in the customer requirements are resolved by the customer and validated again.

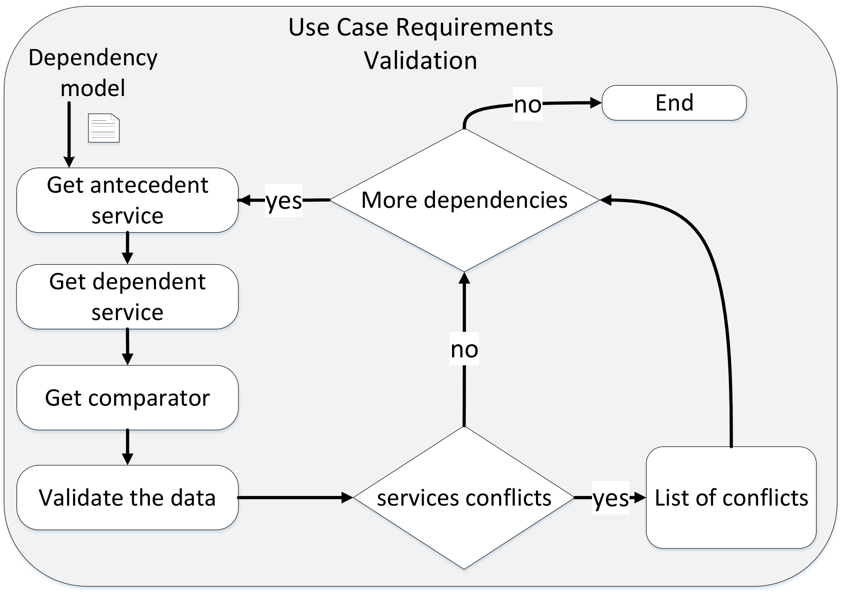


Figure 6.3: Dependency based Use Case requirements validation stages

Based on the Use Case ID, the dependency model of the current requirements is retrieved. From the dependency model the relevant dependencies between each two dependent services are extracted. Each two dependent services are assigned a different dependency ID (i.e., depID) as shown in Listing 1. As a next step, the dependent and antecedent service values are extracted and validated based on the dependency constraint specified for each two dependent relations. If a conflict is detected, the dependent service is added to a list of affected services, which is later displayed to the customer.

6.2.3 Stage (C). Structuring Use Case Requirements

In Use Case requirements, as the number of offered services along with the dependencies between them increases, the Use Case hierarchical structure (shown in Figure 6.1) quickly becomes cluttered and a disordered network of tangled arcs. This makes it hard for (i) the customers to specify their requirements and resolve the conflicts (which requires an expert customer and is time consuming), (ii) the providers to check the dependency relations between their offered services to avoid any violation, and (iii) the providers to identify the critical areas where services should be protected to prevent security breaches. Consequently, the objective of this stage is to embody each Use Case hierarchical structure by mapping the dependencies in a precise order where services are ordered according to their level of dependency (i.e., From the least to the most dependent). This ordering makes the dependency relations explicit and more traceable regardless of the size which allows customers to easily define their security requirements. Furthermore, this provides the Cloud providers with the guidance on the service improvements that should be performed in order to achieve the customer requested service level.

A variety of techniques exist for the analysis, management and ordering of the services other than the graphs used in building the hierarchical structure. One of these techniques is the program evaluation and review technique [WL77]. Although this technique incorporates more information than the directed graphs, it is still inadequate for representing the vast majority of design procedures where iteration[4] task relationships are involved. Another technique which has been widely used in documenting design procedures is the structured analysis and design technique [Ros77]. This technique attempts to overcome the size limitations by restricting the amount of information that can be placed on each document. Unfortunately, loops remain an unsolved problem [MM87].

A representation which overcomes the size and the iteration tasks limitations of those techniques discussed above is the Design Structure Matrix (DSM) [Ste81]. There are two main categories of DSMs: static and time-based [Bro01]. Static DSM represents system elements existing simultaneously, such as components of a product architecture or groups in an organization. In time-based DSM, which is the type of DSM used in this section, the ordering of the rows and columns indicates a flow through time. The DSM embodies the structure of the underlying design activity by mapping the relations between services in a precise order which makes the Use Case requirements clear and easy to read, regardless of the size of the DSM. To clarify, a Use Case of n services is represented as an $n \times n$ matrix with identical row and column labels. The matrix element a_{ij} is empty if the i^{th} column is independent on the j^{th} row, and not empty if they are dependent. This denotes that services with empty rows have all required information and do not depend on others. Furthermore, the empty columns provide no information required by other services.

To demonstrate the idea of DSM, the mapping of the Use Case, shown in Figure 6.1, into a DSM is presented as depicted in Table 6.1. As the dependencies of services on themselves are not considered (as specified earlier in the dependency model), there are no marks along the diagonal. For example, by examining row 2 in Table 6.1 we note that $s1$ depends on $s1.1$, $s1.2$, and $s1.3$.

[4] A loop of information which occurs if there are bidirectional relations between services, which means each service is waiting for information from the other one.

Table 6.1: DSM mapping of UC requirements shown in Figure 6.1

#		**UC**	**s1**	**s2**	**s1.1**	**s1.2**	**s1.3**	**s2.1**	**s2.2**
1	**UC**	.	X	X					
2	**s1**		.		X	X	X		
3	**s2**		X	.		X		X	X
4	**s1.1**				.		X		
5	**s1.2**					.			
6	**s1.3**				X		.		
7	**s2.1**				X			.	
8	**s2.2**					X		X	.

After mapping each Use Case requirments into a DSM, we can start reordering the DSM rows and columns in order to transform the DSM into a lower triangular form (i.e., The matrix has no entries above the diagonal), this is called DSM partitioning [GE91] and is done in two steps:

Step 1: Services which have a minimum number of dependencies (initially there will be none) are placed at the top of the DSM. These services are identified as services with minimum number of row values. If there is more than one such service, the one with maximum number of column values is selected.

Step 2: Services that deliver no information to others in the matrix are placed at the bottom of the DSM. This is easily identified by observing an empty column in the DSM. Once a service is rearranged, it is removed from the DSM and Step 2 is repeated for the remaining elements.

Table 6.2 shows the result of partitioning the DSM depicted in Table 6.1. For example, by examining column 1 in Table 6.2 we note that, $s2.2$, $s1$, and $s2$ depends on $s1.2$. This means that, the violation of $s1.2$ will lead to the violation of $s2.2$, $s1$, and $s2$. Bidirectional dependencies occur when the matrix cannot be reordered to have all matrix elements sub-diagonal. As shown in Table 6.2, $s1.1$ and $s1.3$ are bidirectionally dependent (indicated by light grey shading); $s1.1$ needs the information of $s1.3$ and $s1.3$ needs the information of $s1.1$. If both $s1.1$ and $s1.3$ are regarded as a single composite service, the cycle can be eliminated [SJSJ05].

Table 6.2: Final DSM of UC requirements depicted in Table 6.1 after partitioning and scheduling

		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
		s1.2	**s1.1**	**s1.3**	**s2.1**	**s2.2**	**s1**	**s2**	**UC**
1	**s1.2**	.							
2	**s1.1**		.	X					
3	**s1.3**		X	.					
4	**s2.1**		X		.				
5	**s2.2**	X			X	.			
6	**s1**	X	X	X			.		
7	**s2**	X			X	X	X	.	
8	**UC**						X	X	.

6.3 Evaluation

In this section we apply the presented framework to the four Use Cases specified in the previous chapters. This is performed in progressive phases as follows.

6.3.1 Phase 1: Hierarchical Structure of the Requirements

Based on the analysis of each Use Case requirements specified before, these requirements are modelled using a graphical mapping in a hierarchical structure, as shown in Figures 6.4, 6.5, 6.6 and 6.7. These figures depict UC1, UC2, UC3 and UC4 requirements as specified in Tables 2.1, 3.1, 4.1 and 5.1 respectively.

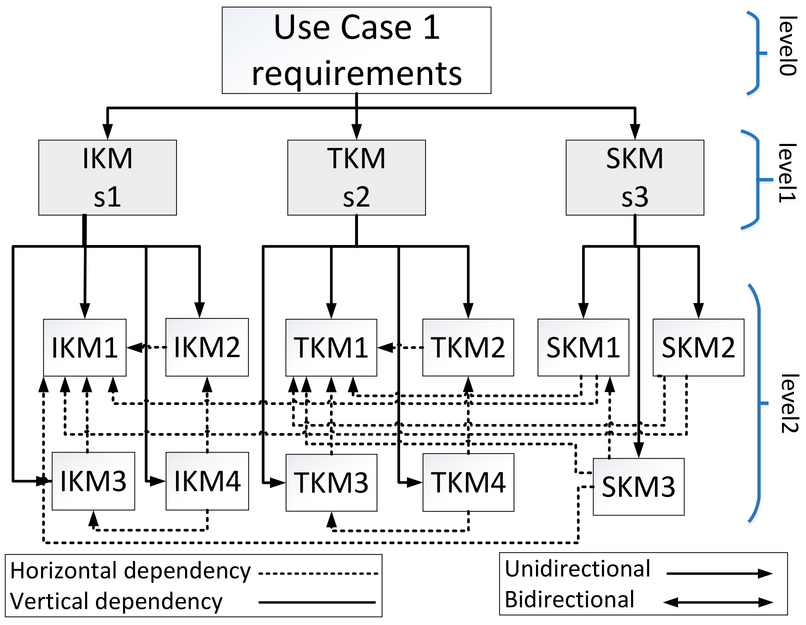


Figure 6.4: Use Case 1 requirements arranged in a hierarchy structure showing the dependencies between requirements

6.3.2 Phase 2: Dependency Model Creation

A dependency model is created for each Use Case to cover all identified dependencies within each Use Case requirements. This model is based on the dependency model introduced in Section 6.2.1.

6.3.3 Phase 3: Use Case Requirements Validation

A meta-model is developed for each Use Case (cf., Section 6.2.2) based on the dependency definitions. This meta-model allows the description of services along with the information on the Use Case requirements drafted for it.

6.3.4 Phase 4: Structuring Use Case Requirements

With the increasing number of offered services and dependencies, the Use Case hierarchical structure (shown in Figures 6.4, 6.5, 6.6 and 6.7) quickly becomes cluttered and a disordered network

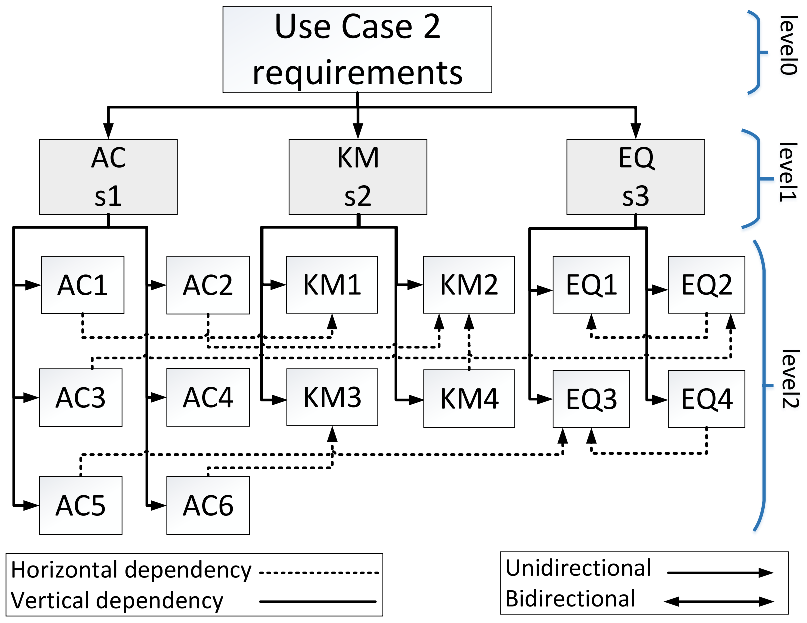


Figure 6.5: Use Case 2 requirements arranged in a hierarchy structure showing the dependencies between requirements

of tangled arcs. The ordering of each Use Case requirements is performed using DSM (cf., Section 6.2.3). To demonstrate the idea of DSM, the mapping of each Use Case shown in Figures 6.4, 6.5, 6.6 and 6.7 into a DSM is presented in this section and is depicted in Tables 6.3, 6.4, 6.5 and 6.6 respectively. As the dependencies of services on themselves are not considered (as specified earlier in the dependency model constrains), there are no marks along the diagonal. For example, by examining:

- Row 2 in the DSM mapping of UC1 requirements (Table 6.3) shows that IKM depends on $IKM1$, $IKM2$, $IKM3$, and $IKM4$.
- Row 3 in the DSM mapping of UC2 requirements (Table 6.4) shows that KM depends on $KM1$, $KM2$, $KM3$, and $KM4$.
- Row 13 in the DSM mapping of UC3 requirements (Table 6.5) shows that $AC7$ depends on $KM1$ and $AC1$.
- Rows 9 and 10 in the DSM mapping of UC4 requirements (Table 6.6) shows that $AC5$ and $AC6$ $AC5$ are bidirectionally dependent .

After mapping each Use Case requirement into a DSM, we can start reordering the DSM rows and columns in order to transform the DSM into a lower triangular form as detailed in Section 6.2.3.

Tables 6.7, 6.8, 6.9 and 6.10 show the result of partitioning the DSM depicted in Tables 6.3, 6.4, 6.5 and 6.6 respectively. Bidirectional dependencies occur when the matrix cannot be reordered to have all matrix elements sub-diagonal. As shown in Table 6.10, $SS1$ and $SS3$ are bidirectionally dependent (indicated by light shading); $SS1$ needs the information of $SS3$ and $SS3$ needs the information of $SS1$. If $SS1$ and $SS3$ are regarded as a single composite service, the cycle can be eliminated [SJSJ05]. Similarly, as shown in Table 6.10, $AC5$ and $AC6$ are bidirectionally dependent, as well as $DE2$, $DE3$, $DE1$ and $SS5$.

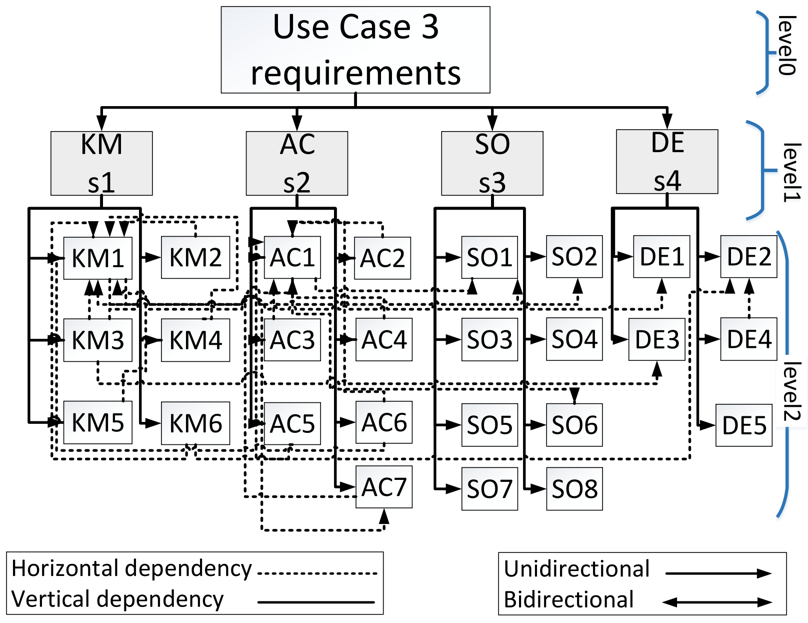


Figure 6.6: Use Case 3 requirements arranged in a hierarchy structure showing the dependencies between requirements

Consequently, on a per Use Case basis Tables 6.7-6.10 outline all possible direct and indirect dependencies across requirements. This dependency structuring enables the customer to identify, quantify, and address the security risks associated with his/her requirements .

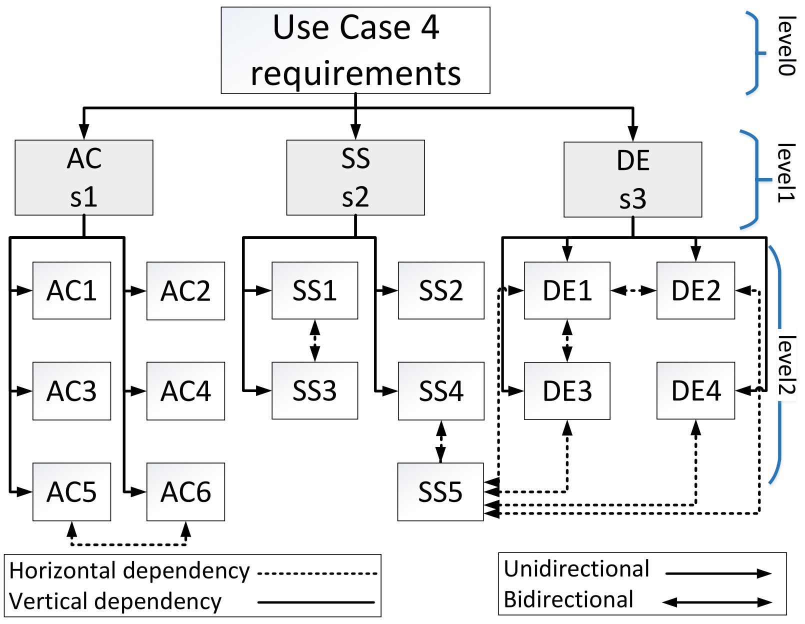


Figure 6.7: Use Case 4 requirements arranged in a hierarchy structure showing the dependencies between requirements

Table 6.3: DSM mapping of UC1 requirements shown in Figure 6.4

#		UC1	IKM	TKM	SKM	IKM1	IKM2	IKM3	IKM4	TKM1	TKM2	TKM3	TKM4	SKM1	SKM2	SKM3
1	**UC1**	.	X	X	X											
2	**IKM**		.			X	X	X	X							
3	**TKM**			.						X	X	X	X			
4	**SKM**				.									X	X	X
5	**IKM1**					.										
6	**IKM2**					X	.									
7	**IKM3**					X		.								
8	**IKM4**						X	X	.							
9	**TKM1**									.						
10	**TKM2**									X	.					
11	**TKM3**									X		.				
12	**TKM4**										X	X	.			
13	**SKM1**					X				X				.		
14	**SKM2**					X				X					.	
15	**SKM3**					X				X				X		.

Table 6.4: DSM mapping of UC2 requirements shown in Figure 6.5

#		UC2	AC	KM	EQ	AC1	AC2	AC3	AC4	AC5	AC6	KM1	KM2	KM3	KM4	EQ1	EQ2	EQ3	EQ4
1	**UC2**	.	X	X	X														
2	**AC**		.			X	X	X	X	X	X								
3	**KM**			.								X	X	X	X				
4	**EQ**				.											X	X	X	X
5	**AC1**					.						X							
6	**AC2**						.						X						
7	**AC3**							.									X		
8	**AC4**								.										
9	**AC5**									.								X	
10	**AC6**										.			X					
11	**KM1**											.							
12	**KM2**												.						
13	**KM3**													.					
14	**KM4**												X		.				
15	**EQ1**															.			
16	**EQ2**															X	.		
17	**EQ3**																	.	
18	**EQ4**																	X	.

Table 6.5: DSM of excerpt of UC3 requirements shown in Figure 6.6

#		UC3	KM	AC	SO	DE	KM1	KM2	KM3	KM6	AC1	AC2	AC6	AC7	SO1	SO2	SO6	DE1	DE2	DE3
1	**UC3**	▪	X	X	X	X														
2	**KM**		▪				X	X	X	X										
3	**AC**			▪							X	X	X	X						
4	**SO**				▪										X	X	X			
5	**DE**					▪												X	X	X
6	**KM1**						▪							X	X	X				
7	**KM2**						X	▪												
8	**KM3**						X		▪									X		X
9	**KM6**						X			▪									X	
10	**AC1**										▪				X		X			
11	**AC2**										X	▪								
12	**AC6**						X				X		▪							
13	**AC7**						X				X			▪						
14	**SO1**														▪					
15	**SO2**															▪				
16	**SO6**																▪			
17	**DE1**																	▪		
18	**DE2**																		▪	
19	**DE3**																			▪

Table 6.6: DSM mapping of UC4 requirements shown in Figure 6.7

#		**UC4**	**AC**	**SS**	**DE**	**AC1**	**AC2**	**AC3**	**AC4**	**AC5**	**AC6**	**SS1**	**SS2**	**SS3**	**SS4**	**SS5**	**DE1**	**DE2**	**DE3**	**DE4**
1	**UC4**	.	X	X	X															
2	**AC**		.			X	X	X	X	X	X									
3	**SS**			.								X	X	X	X	X				
4	**DE**				.												X	X	X	X
5	**AC1**					.														
6	**AC2**						.													
7	**AC3**							.												
8	**AC4**								.											
9	**AC5**									.	X									
10	**AC6**									X	.									
11	**SS1**											.		X						
12	**SS2**												.							
13	**SS3**											X		.						
14	**SS4**														.	X				
15	**SS5**														X	.	X	X	X	X
16	**DE1**															X	.	X	X	
17	**DE2**															X	X	.		
18	**DE3**															X	X		.	
19	**DE4**															X				.

Table 6.7: Final DSM of UC1 requirements depicted in Table 6.3 after partitioning and scheduling

		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
		IKM1	**TKM1**	**IKM2**	**IKM3**	**TKM2**	**TKM3**	**IKM4**	**SKM1**	**SKM2**	**TKM4**	**SKM3**	**SKM**	**IKM**	**TKM**	**UC1**
1	**IKM1**	.														
2	**TKM1**		.													
3	**IKM2**	X		.												
4	**IKM3**	X			.											
5	**TKM2**		X			.										
6	**TKM3**		X				.									
7	**IKM4**			X	X			.								
8	**SKM1**	X	X						.							
9	**SKM2**	X	X							.						
10	**TKM4**					X	X				.					
11	**SKM3**	X	X						X			.				
12	**SKM**								X	X		X	.			
13	**IKM**	X		X	X			X						.		
14	**TKM**		X			X	X				X				.	
15	**UC1**												X	X	X	.

Table 6.8: Final DSM of UC2 requirements depicted in Table 6.4 after partitioning and scheduling

		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
		AC4	**KM1**	**KM2**	**KM3**	**EQ1**	**EQ3**	**AC1**	**AC2**	**KM4**	**AC6**	**AC5**	**EQ2**	**EQ4**	**AC3**	**KM**	**EQ**	**AC**	**UC2**
1	**AC4**	.																	
2	**KM1**		.																
3	**KM2**			.															
4	**KM3**				.														
5	**EQ1**					.													
6	**EQ3**						.												
7	**AC1**		X					.											
8	**AC2**			X					.										
9	**KM4**			X						.									
10	**AC6**				X						.								
11	**AC5**						X					.							
12	**EQ2**					X							.						
13	**EQ4**						X							.					
14	**AC3**												X		.				
15	**KM**		X	X	X					X						.			
16	**EQ**					X	X						X	X			.		
17	**AC**	X						X	X		X	X						.	
18	**UC2**															X	X	X	.

Table 6.9: Final DSM of excerpt of UC3 requirements depicted in Table 6.5 after partitioning and scheduling

		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**
		SO1	**SO2**	**SO6**	**DE1**	**DE2**	**DE3**	**SO**	**DE**	**AC1**	**AC2**	**KM1**	**AC7**	**KM2**	**KM6**	**AC6**	**KM3**	**KM**	**AC**	**UC3**
1	**SO1**	.																		
2	**SO2**		.																	
3	**SO6**			.																
4	**DE1**				.															
5	**DE2**					.														
6	**DE3**						.													
7	**SO**	X	X	X				.												
8	**DE**				X	X	X		.											
9	**AC1**	X		X						.										
10	**AC2**									X	.									
11	**KM1**	X	X									.	X							
12	**AC7**									X		X	.							
13	**KM2**											X		.						
14	**KM6**											X			.					
15	**AC6**									X		X				.				
16	**KM3**				X		X					X					.			
17	**KM**											X		X	X		X	.		
18	**AC**									X	X		X			X			.	
19	**UC3**							X	X									X	X	.

Table 6.10: Final DSM of UC4 requirements depicted in Table 6.6 after partitioning and scheduling

		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**
		AC1	**AC2**	**AC3**	**AC4**	**SS2**	**SS1**	**SS3**	**AC5**	**AC6**	**DE2**	**DE3**	**DE1**	**SS5**	**SS4**	**DE4**	**AC**	**SS**	**DE**	**UC4**
1	**AC1**	.																		
2	**AC2**		.																	
3	**AC3**			.																
4	**AC4**				.															
5	**SS2**					.														
6	**SS1**						.	X												
7	**SS3**						X	.												
8	**AC5**								.	X										
9	**AC6**								X	.										
10	**DE2**										.		X	X						
11	**DE3**											.	X	X						
12	**DE1**										X	X	.	X						
13	**SS5**										X	X	X	.	X	X				
14	**SS4**													X	.					
15	**DE4**													X		.				
16	**AC**	X	X	X	X				X	X							.			
17	**SS**					X	X	X						X	X			.		
18	**DE**										X	X	X			X			.	
19	**UC4**																X	X	X	.

6.4 Identifying Violations

In order to understand the violations that could occur between the services, we first have to define and measure the risks causing these violations and also their severity. Based on the assumed impact and threat severity of the Use Cases as specified in the previous chapters (cf., Tables 2.2, 3.2, 4.2 and 5.2), the concept of risk is defined as a measure of the expected negative effect of a particular unwanted event (i.e., Assumed Impact) [JW98]. This is expressed as the product of the *Violation Likelihood* of the event and the expected damage (*Threat Severity*). This measure is then used to determine if these risks are acceptable and to decide which ones to mitigate first.

On a per Use Case basis, Tables 6.7, 6.8, 6.9, and 6.10 outline all possible direct and indirect dependencies across the requirements. Furthermore, the tables enumerate the service requirements as ordered by their criticality. This DSM-based process also ensures completeness of ascertaining the dependencies to the level of information available in the requirements by categorizing the marked X's. In the following, we show an example sampling of the possible Use Case violations as identifiable from the developed DSM tables corresponding to the different Use Cases.

- **UC1:** As depicted in Table 6.7 column 1, the violation of *IKM1* will lead to the violation of *IKM2*, *IKM3*, *SKM1*, *SKM2*, *SKM3*, and *IKM*. Similarly, the violation of *TKM1* directly affects *TKM2*, *TKM3*, *SKM1*, *SKM2*, *SKM3*, and *TKM* as depicted in Table 6.7 column 2. This is expected as breaching the CRUD operations for infrastructure keys leads to the violation of policy-driven support for standard APIs and protocols in infrastructure-key management. Furthermore, the breaching the CRUD operations for tenant keys leads to the violation of policy-driven support for standard APIs and protocols in tenant-key management.

- **UC2:** The breach of *EQ3* leads to the breach of *AC5*, *EQ4*, and *EQ*. For example, this is the case if a user has access to data which was supposed to be revoked, resulting in a partial confidentiality violation. Also, the violation of *KM1* directly affects *AC1* as shown in Table 6.8 column 2. This means that if the keys are compromised and thus one key per client requirement is violated, then the access control per client requirement is violated as well. We also note that by violating the confidentiality of group data, an attacker can obtain access per group of clients, and derive the group keys as depicted in Table 6.8 column 3.

- **UC3:** *KM1* violation leads to the violation of *AC7*, *KM2*, *KM6*, *AC6*, *KM3*, and *KM* as depicted in Table 6.9 column 11. This means that if an attacker can get an instance of a key management service provisioned to a tenant from the Cloud service, then the attacker will be able to generate, insert, retrieve and remove keys from the key management service. Furthermore, the attacker is able to cache the keys on trusted virtual machines or gateways.

- **UC4:** The breach of *SS5* compromises *DE1*, *DE2*, *DE3*, *DE4*, and *SS4* as depicted in Table 6.10 column 13. Therefore, if there are no appropriate technical and organizational measures implemented by the Cloud provider, the customer's personal data is susceptible to unauthorized access, disclosure or alteration. Furthermore, each user should have a limit of attempts to access the system in order to prevent brute force attempts. Thus, if a user introduces multiple wrong credentials, he will be locked. Consequently, only the administrator of the system (the ESCUDO-CLOUD middleware) should be able to enable the access of the locked user as depicted in Table 6.10 columns 8 and 9.

6.5 Summary

Overall, we have been able to achieve the following;

- Develop an analytical process that is able to systematically capture both direct and indirect dependencies at the level of the Use Case requirements.
- Assess the degree of dependencies across the requirements.
- Validate the effectiveness of a requirements based analytical process.
- Identify potential threats that could occur for the ESCUDO-CLOUD Use Cases, and specify the critical areas where services should be protected.

The developed process is a highly innovative contribution that was initially presented in [TMT$^+$16], and has also been generalized to apply beyond the ESCUDO-CLOUD Use Cases provided that the requirements capture is available.

The next chapter details the threat analysis covering the associated Use Cases presented in the prior chapters. This involves utilizing information from the Use Cases on the requirement references, the number of requirements, the maximum/average values for the *Violation Likelihood* and *Threat Severity*, and the DSM ordering of the requirements from the least to the most dependent services. These *Violation Likelihood* and *Threat Severity* measurements as well as the violations identification (presented in this chapter) for each Use Case are considered to illustrate the violations severity that could potentially occur across the services.

7. Summary Analysis from Use Cases and Threat Analysis

The previous sections of this document have defined the use case requirements and provided risk assessments. Section 6 aggregated and analysed these requirements to develop dependancy models across the requirements to more accurately identify risks that require higher priority when implementation mitigation measures. This section continues the analysis of the requirement sets and summarizes the logical categorization of Use Case requirements. It also presents three visualizations of the requirements data from the Use Cases. It examines the *Violation Likelihood* and *Threat Severity* measurements to highlight the critical areas where security mechanisms should be applied to protect the services. The requirements are grouped initially in Table 7.1 according to the categories defined in D2.1, providing maximum as well as average figures. Figure 7.1 illustrates the global distribution of the requirements on the axes of *Violation Likelihood* and *Threat Severity* while Figure 7.2 presents the data across each Use Case individually, capturing a view of the Use Cases that clearly identifies in which Use Cases the highest threat concerns exist.

7.1 Common Requirements Catalogue

Table 7.1 summarizes the requirements outlined for each of the Use Cases presented in Chapters 2-5. Here, the common requirements are grouped together into logical categories. This distribution of requirements highlights the key areas for the different Use Cases. The requirement descriptions used to categorize the requirements correspond to the descriptions provided in D1.2 (Sections 4.2.1 to 4.2.10).

The table indicates for each requirement category, the associated Use Cases, the requirement references, the number of requirements, and the maximum and average values for the *Violation Likelihood* and *Threat Severity* based on the data presented in the previous chapters. While the maximum value can identify which category contains the most critical individual requirement in terms of violation likelihood and threat severity, the average values quickly indicate which categories present a higher degree of risk.

A facet of the statistics provided in Table 7.1 which must be taken into consideration is the number of requirements within each category. For example, the category ”Data confidentiality through access control policies” contains thirty-two requirements, whereas the category ”Transmission over secure channels” contains only two.

The effect of such a disparity in the number of per-category requirements is that the number of requirements can effectively be viewed as a weighting; the average violation likelihood statistic of the ”Data confidentiality through access control policies” is 3.37, slightly less than the ”Transmission over secure channels” category’s 3.5. Consequently, despite having a lower average likelihood violation statistic, the ”Data confidentiality through access control policies” category would be more prone to such a violation occurring simply due to the greater volume of requirements it contains.

The concept of the number of requirements in a category acting as a weight is also valid in the context of a scenario regarding threat severity; thirty-two low-severity threats could accumulatively pose a greater threat than two high-severity threats.

In terms of aggregated requirements, four categories contain a high level of coverage with >20 requirements in each:

- Data confidentiality through access control policies **[32 requirements]**
- Controlling data modification via key manager and access controller components **[20 requirements]**
- Controlling availability of data **[20 requirements]**
- Data owners allocated key manager and access controller **[23 requirements]**

This is a strong indication of the anticipated risks identified within the area of controlling access to the data as each of the highlighted categories deals with either access control or key management.

Other categories featuring a moderate number of requirements are:

- Scalability of the infrastructure does not compromise data confidentiality **[10 requirements]**
- Fine-grained data access **[9 requirements]**
- Federated-Cloud architectures **[11 requirements]**

The indication here again points to an emphasis on access control to protect the confidentiality of data. The *Federated Cloud architectures* category also includes several key management requirements and an access control requirement.

With respect to the weights assigned to the violation likelihood and threat severity in each category, a quick observation of Table 7.1 highlights some key areas that indicate both that a violation in this area is likely but also that the threat posed by such a violation is high.

- *Key hierarchies for fine grained data access*
- *Secure transfer of data, control messages and keys*
- *Data confidentiality through access control policies*
- *Secure deletion of data by deleting key*

However, these values mask the detail of each specific requirement. For example, within *Key hierarchies for fine grained data access*, REQ-UC2-KM1 and REQ-UC2-KM4 have a high level of threat severity, however the likelihood of a violation is low. Only REQ-UC2-KM2 (*Group key management*) registers a high score for violation likelihood and a moderate threat severity within this category.

Requirement Description	**Use Case #**	**Requirement Reference**	**# of Requirements**	**Max Violation Likelihood**	**Average Violation Likelihood**	**Max Threat Severity**	**Average Threat Severity**
Data confidentiality through access control policies	UC1 UC2 UC3 UC4	REQ-UC1-IKM-***[1-2]***, REQ-UC1-TKM-***[1-2]***, REQ-UC2-AC-***[1-6]***, REQ-UC2-KM-***[1-2,4]***, REQ-UC2-EQ-4, REQ-UC3-KM-***[1-5]***, REQ-UC3-AC-***[1-7]***, REQ-UC3-SO-1, REQ-UC3-DE-1, REQ-UC4-AC-***[1-2,4]***, REQ-UC4-DE-1	32	7	3.37	10	6.06
Appropriate levels of encryption	UC2	REQ-UC2-EQ-***[1-3]***	3	4	3.33	10	5.33
Secure deletion of data by deleting key	UC1	REQ-UC1-IKM-4, REQ-UC1-TKM-4	2	6	6	10	10
Scalability of the infrastructure does not compromise data confidentiality	UC1 UC3	REQ-UC1-SKM-***[2-3]***, REQ-UC3-KM-6, REQ-UC3-DE-***[2,4-5]***, REQ-UC4-DE-***[2-4]***	10	5	2.4	10	5.1
Data confidentiality should comply with relevant standards	UC2 UC4	REQ-UC2-KM-3, REQ-UC4-SS-***[4-5]***	3	2	2	7	6
Controlling data modification via key manager and access controller components	UC2 UC3	EQ-UC2-AC-***[5-6]*** REQ-UC2-KM-***[1-2,4]***, REQ-UC3-KM-***[2,4-6]***, REQ-UC3-AC-***[1-5,7]***, REQ-UC3-SO-1, REQ-UC3-DE-***[1-2,4-5]***	20	8	3.1	10	5.7

Encryption standards	UC2 UC3	REQ-UC2-KM-3, REQ-UC3-KM-3, REQ-UC3-AC-6, REQ-UC3-DE-3	4	5	2.75	8	5.5
Transmission over secure channels	UC1	REQ-UC3-DE-5, REQ-UC4-SS-5	2	5	3.5	8	7
Controlling availability of data	UC1 UC3 UC4	REQ-UC1-IKM-***[1-2]***, REQ-UC1-TKM-***[1-2]***, REQ-UC3-KM-***[1-5]***, REQ-UC3-AC-***[2-5]***, REQ-UC3-SO-1, REQ-UC3-DE-***[2-4]***, REQ-UC4-AC-6, REQ-UC4-SS-2, REQ-UC4-DE-4	20	7	2.95	10	7.3
Resilience of encryption keys	UC1 UC3 UC4	REQ-UC1-SKM-1, REQ-UC3-DE-5, REQ-UC4-AC-5, REQ-UC4-SS-5	4	5	3	8	6.75
Scaling security framework	UC1 UC3	REQ-UC1-SKM-***[2-3]***, REQ-UC3-KM-6, REQ-UC3-AC-***[1,6,7]***, REQ-UC3-DE-1	7	5	3.42	10	5.57
Data owners allocated key manager and access controller	UC1 UC2 UC3 UC4	REQ-UC1-IKM-1, REQ-UC1-TKM-1, REQ-UC2-KM-***[1,3]***, REQ-UC2-AC-***[3-4]***, REQ-UC3-KM-***[1-2,4-5]***, REQ-UC3-AC-***[1-2,4-5]***, REQ-UC3-SO-1, REQ-UC3-DE-***[1,3]***, REQ-UC4-AC-***[1-2,5-6]***, REQ-UC4-SS-***[2,5]***, REQ-UC4-DE-1	23	7	2.86	10	5.78
Point of data encryption	UC1 UC3	REQ-UC4-DE-***[2-4]***	3	1	1	8	5.66
Key hierarchies for fine grained data access	UC2	REQ-UC2-KM-***[1-2,4]***	3	8	4.33	10	8.33
Managing multiple users through groups and roles	UC2 UC4	REQ-UC2-AC-***[2,4-5]***,REQ-UC4-AC-4	4	4	3.25	4	1.75

Secure transfer of data,control messages and keys	UC1 UC3 UC4	REQ-UC1-IKM-1, REQ-UC1-TKM-1, REQ-UC3-KM-2, REQ-UC4-SS-2, REQ-UC4-DE-4	5	7	4	10	8
Fine-grained data access	UC1 UC2 UC3 UC4	REQ-UC1-IKM-2, REQ-UC1-TKM-2, REQ-UC2-AC-3, REQ-UC2-AC-4, REQ-UC2-EQ-4, REQ-UC3-AC-7, REQ-UC4-DE-***[1-3]***	9	5	3.33	9	5.11
Multi-layered encryption schemas	UC2	REQ-UC2-EQ-***[1-3]***	3	4	3.33	10	5.33
Write operations with different keys	UC2	REQ-UC2-EQ-4	1	5	5	2	2
Dynamic CSP storage expansion	UC4	REQ-UC4-SS-***[1,3]***	2	3	2.5	6	6
Protection against data overwrites	UC4	REQ-UC4-SS-5	1	2	2	6	6
Caching user's credentials on write operations	UC4	REQ-UC4-AC-2	1	3	3	8	8
Key management service supports open standards	UC1	REQ-UC1-IKM-3, REQ-UC1-TKM-3	2	5	5	10	10
Unique user credentials within CSP	UC3 UC4	REQ-UC3-SO-1, REQ-UC4-SS-5	2	2	2	6	4
Federated-Cloud architectures	UC3 UC4	EQ-UC3-KM-***[1-2,5-6]***, REQ-UC3-AC-1, REQ-UC3-SO-***[2,6-8]***, REQ-UC4-SS-***[2,5]***	11	4	2.27	10	5.54

Table 7.1: Common Use Case requirements summary

7.1.1 Interpreting the Requirement Correlations

In order to provide a more fine grained overview of the requirements, Figures 7.1 and 7.2 plot every requirement against their respective *Violation Likelihood* and *Threat Severity* values. Where more than one value was provided for a requirement (e.g. Table 2.2: REQ-UC1-IKM-1), the highest occurring values for *Violation Likelihood* and *Threat Severity* were used to plot the figures. A clear pattern is visible in Figure 7.1 whereby very few requirements are believed at risk of a violation. Only five requirements were identified as having a *Violation Likelihood* value greater than five (the maximum possible value being ten). As inferred earlier from Table 7.1, the requirements with the greatest risk are in the key management area (notably in UC1 and UC2).

Figure 7.2 provides a per-use-case view of *Violation Likelihood* and *Threat Severity* which allows for certain insights to be made. The values of likelihood and severity on the axes increase with proximity to the middle of the graph.

- The *Threat Severity* values for UC1 are all above five, indicating a high level of impact should a threat become realized; the likelihood of such a violation occurring is low-to-moderate, with the distribution being even with a high point of seven. UC1 deals with data at rest encryption as well as the protection and authenticity of stored data; virtually any violation in this area would have critical consequences, such as exposure of tenant data or key information leakage, hence the aforementioned assignment of high *Threat Severity* scores.

- With respect to UC2, there is a consistent distribution of points from low to high values for *Threat Severity*. The distribution for *Violation Likelihood* is almost entirely between two and five, with the exception of one score of eight. UC2 concerns secure enterprise data management in the cloud. The even distribution of points is, of course, equivalent to their respective requirements which range from low and moderate violation impacts such as 'user inconvenience' and 'partial confidentiality violation'. Points on the upper extremity of the severity scale have more impact, for example: 'keys guessable' and 'global confidentiality violation'.

- The distribution of points for UC3 and UC4 in Figure 7.2 are quite similar; in both cases the *Violation Likelihood* values are of five or lower, however there is a disparity between the two use cases in terms of the *Threat Severity*; the values are relatively even in distribution with regards to UC3, but for UC4 the majority of *Threat Severity* points are in excess of five. UC3 deals with federated secure cloud storage; examples of the most severe impacts of threats are confidentiality and integrity violations of keys and tenant data, as well as loss of availability (denial of service). UC4 addresses elasticated cloud service provisioning, and shares similar high impact threat to those relevant to UC3 (data confidentiality and integrity violation, total access to storage systems), though there are fewer threats identified for UC4 in comparison to UC3.

While the risk of a violation may not be considered highly likely for the majority of the requirements, the severity of the threat posed by any violation reduces the limits of what is deemed an acceptable risk. Of the sixty six requirements across the four Use Cases, thirty register a threat severity greater than eight which would indicate a complete, or near complete, compromise of the environment. Even if the likelihood of a violation for these is low, the prospect of such a catastrophic failure should ensure these requirements are adequately considered.

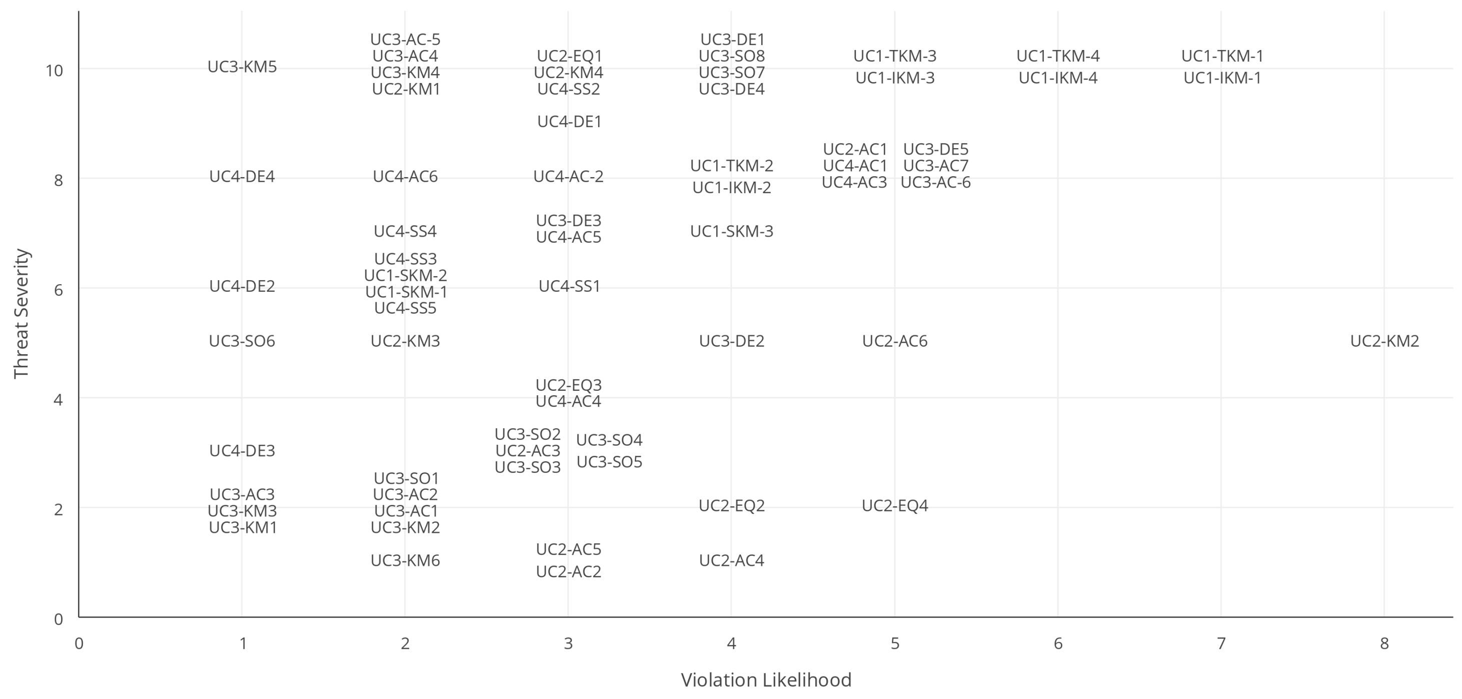


Figure 7.1: Requirements (Violation Likelihood vs Threat Severity)

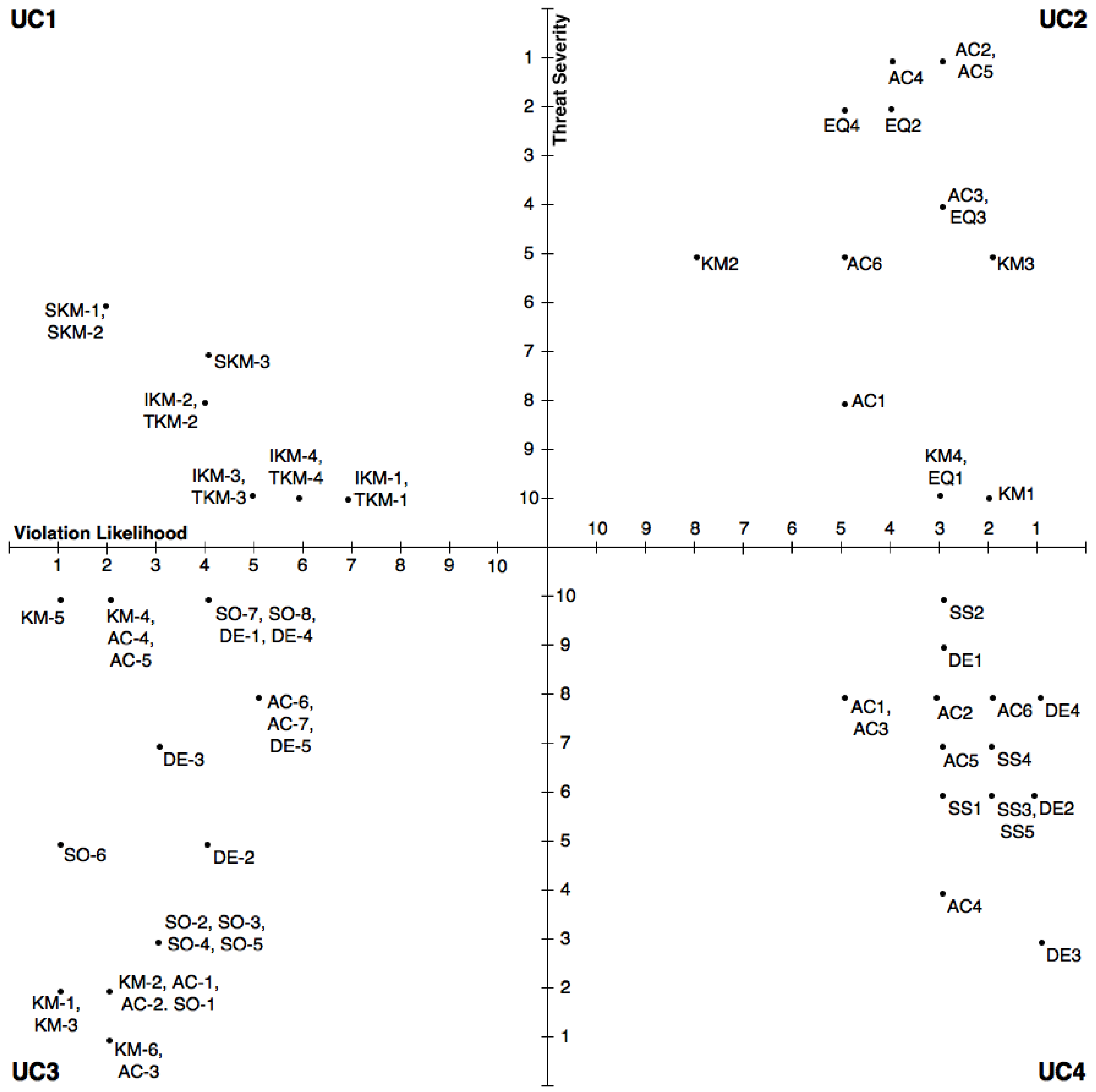


Figure 7.2: Requirements (Heat Map by Use Case)

7.1.2 Summary Insights

One insight taken from the summary analysis with regards to UC1 (storage and key management techniques for cloud platforms) concerns its high severity rating, proportionately speaking; six of the eleven requirements are assigned a rating of ten. Interestingly these six requirements mirror each other: REQ-UC1-IKM[1-3] are duplicates of REQ-UC1-TKM[1-3], with the obvious exception of the former targeting infrastructure key management and the latter targeting tenant key management; this is evident from the aforementioned requirements being paired at coordinates in Figure 7.2. The severity of a violation of one of these requirements is compounded by the moderate likelihood of its occurrence, which is quite a useful observation of the threat analysis' data.

A second insight gained from the summary analysis is that the severity rating of UC4 (Elastic Cloud Service) is characterized by the cluster of access-control requirements located on severity ratings seven and eight of Figure 7.2. These requirements centre around user authentication typically found in a login system (limiting failed attempts, saving credentials to the device). The impacts of violations for these requirements are very similar, or even identical, hence the positional clustering. The combined impact of these severity ratings, along with some notably higher ratings assigned to REQ-UC4-SS2 (storage access control through middleware) and REQ-UC4-DE1 (only the data owner can encrypt the data), is sufficient for the overall severity to be considered moderate rather than low. Given the generally low likelihood of a violation coupled with the aforementioned moderate severity rating; UC4 exhibits a strong overall performance in the threat analysis.

In the next chapter a tool based architectural threat modeling approach is presented. The tool provides a partly automatic security assessment. This assessment is based on (i) the threat analysis covered in the previous chapters and the system required security and (ii) the high-level design enriched with security aspects.

8. Automation Techniques for Threat Analysis

Within the previous chapters a Use Case centric analysis of threats was performed. In contrast, in this chapter we want to contribute and illustrate a novel approach towards automatic threat management that is not limited to the ESCUDO-CLOUD Use Cases. Threat modeling at the conceptual design level is a promising field of research and is already supported by several methodologies, such as STRIDE [HLOS06]. The primary task is to identify potential threats and vulnerabilities based on assets, security objectives, and information available from various architectural design documents. As could be inferred from the previous sections of this deliverable a drawback of current approaches is that they require significant manual work to identify and assess potential threats. Additionally, the analysis often lacks focus on the primary assets that require protection. In the following we formulate an architectural threat modeling approach supported by a tool that allows partly automating and focusing security assessment based on (1) assets and security objectives in the system; (2) high-level design enriched with security aspects. By complementing architecture diagrams with security-related information a lightweight (automated) threat identification is enabled. Besides, the information captured during this analysis can serve as a good high-level documentation of the threat modeling results.

8.1 Motivation

In the current era of ubiquitous computing, the problem of building secure software in the Cloud is gaining an increased attention. Every year thousands of critical software vulnerabilities are discovered and exploited [NVDa, CER]. As a result of these attacks software vendors lose credibility, the privacy of consumers is compromised, their bank accounts are cleared. The majority of security incidents neither happen at the network nor the operating system level but at the application level, i.e. due to insecure application software [VM01, XN06].

In the development process software passes through several stages: from requirements analysis and architectural design to documentation and maintenance. Security vulnerabilities can be introduced in a system not only during the implementation phase but as early as in the requirements analysis and architectural design phases. In fact, Microsoft reported in 2002 that about 50% of the problems are found to be at the design level [HM04]. Not only are vulnerabilities in the software architectural design frequent, they are also much more costly to fix. Table 8.1 from [HL06] shows the relative costs of fixing defects introduced at different points in the Software Development Lifecycle (SDL).

In simple economic terms, finding and removing bugs in a software system before its release is orders of magnitude cheaper and more effective than trying to fix them afterwards [VM01]. Therefore, it is vital that the adopted software development process incorporates the best security practices throughout its lifecycle, especially in its starting phases [McG06, p. 154].

A common practice is performing the risk assessment of the architectural design, based on system assets and various design specifications. The solutions range from ad-hoc approaches, based

Defect Introduction Point	Defects Found During Requirements	Defects Found During Architecture	Defects Found During Construction	Defects Found During Test	Defects Found After Release
Requirements	1	3	5-10	10	10-100
Architecture	None	1	10	15	25-100
Construction	None	None	1	10	10-25

Table 8.1: Relative cost of removing software defects

on brainstormings, through more structured, yet still informal, to rigorous, based on mathematical models and model checking (see Section 8.2). Formal approaches provide powerful reasoning capabilities but require great expertise for operation, which is why their use for risk assessment is mostly academical. Industry, however, demands practical techniques feasible for use by security analysts, architects and regular developers.

STRIDE threat modeling [LH05, HLOS06] is a practical method successfully used in industrial context: an informal, yet systematic, approach to assess architecture diagrams against threats and estimate their risks. After analyzing STRIDE and other threat modeling methods (see Section 8.2) we concluded that they are missing the following:

1. A precise reusable approach to describe the security aspects of an application (e.g. the architectural design and security-related information are often expressed as informal diagrams with text annotations [Mei03, Bor11]).

2. An automated support for the identification of potential threats and concrete follow-ups.

3. A comprehensive system view: consideration of all software-centric, attacker-centric and asset-centric views [Sho].

In this chapter we sought how to automatically support the elicitation of conceptual design-level threats and focus the analysis on the most important issues. This, overall, should make the threat modeling work more efficient. We build upon the work of Borozdin [Bor11] enhancing STRIDE in the following directions:

1. Enrich the architectural design with some additional security aspects (access permissions, security controls and trust zones) and validate architecture diagrams.

2. Support threat identification by performing interaction analysis on architecture diagrams to model the propagation of possible attacks violating stated security objectives.

3. Consider asset-centric and attacker-centric views in the analysis: provide a high-level risk overview of an application containing the stakeholders, security objectives, possible threat sources and risks.

In addition, we realized the proposed ideas in a software prototype Tam^2 (a successor of the tool developed by Borozdin [Bor11]).

8.1.1 ESCUDO-CLOUD innovation

The work in this chapter bases on several state of the art solutions and established standards (CORAS [LSS11], STRIDE [HLOS06], FMC/TAM [TAMb]) to provide a more effective architectural threat modeling. Besides, we verified that even little additional security-related information on the architecture diagrams can enable a lightweight guided security analysis. To achieve this we introduced a new technique to automatically identify threats to security objectives by analyzing the interaction between architectural components.

8.2 State of the Art

This section provides an overview of the literature related to security risk analysis methods, which can be used in the requirements and design phases of software development.

STRIDE One prominent threat modeling approach, called STRIDE [HLOS06], was introduced by Microsoft as part of its Trustworthy Computing Security Development Lifecycle [Lip04]. The name comes from the used threat classification scheme (**S**poofing, **T**ampering, **R**epudiation, **I**nformation Disclosure, **D**enial of Service and **E**levation of Privileges). The STRIDE process consists of decomposing an application into constituent components on data flow diagrams and analyzing each component against possible threats from different STRIDE categories, and mitigating the identified risks. The application design is represented as a Data Flow Diagram (DFD) consisting of Data Flows, Data Stores, Processes, Interactors and Trust Boundaries.

In practice, security engineers performing a STRIDE-based threat assessment will build a data-flow diagram of the system and start asking leading questions about all diagram components. Answering those questions may reveal certain application vulnerabilities and prompt their elimination. This technique is usually followed by a risk estimation procedure, such as CVSS [CVS], Microsoft's Severity Rating System [Micb] or Exploitability Index [Mica].

Though practical, STRIDE has some shortcomings that limit its effective usage:

- Any predefined checklist of possible attacks will be incomplete and can miss new creative attacks [McG06, p. 147].
- Composition of individual secure components is not necessarily secure. As pointed in [HLOS06] and [McG06, p. 159], threats often emerge only when the components are assembled together to create a system.
- The process is labor intensive with most work being performed manually (usually during a design security review session). Existing tools mainly guide this process, but they do not perform any automatic reasoning on architecture diagrams.
- This approach is software-centric, i.e. focuses on software during the threat modeling. It does not consider neither the most important information assets (asset-centric view) — like most of other risk analysis approaches do — nor possible attacker capabilities and motivations.

Microsoft developed two tools that support the STRIDE process: SDL threat modeling tool [SDL] and TAM tool [TAMa].

SDL threat modeling tool provides drawing functionality for DFD diagrams, as well as their validation. It also understands the notion of trust boundaries and thereby suggests application entry points that should be secured in the first place. Unfortunately, this is where the reasoning capabilities of the tool end and the remaining features are related mostly to the documentation of the threat modeling process. The tool also lacks the functionality to specify the security objectives and information assets.

Threat Analysis and Modeling Tool (TAM) uses a top-down approach to document application threat models. The workflow starts with specifying application requirements (business objectives, roles, etc) and application design (components, data, access control matrix, etc.). Based on the provided information some possible threats to Confidentiality, Integrity and Availability are identified. The tool has an attack library that captures knowledge on possible attacks and countermeasures. Lastly, it provides means to estimate the risk levels of identified threats and prioritize them. The tool does not support drawing architecture diagrams, does not take into account existing countermeasures and does not allow to specify security objectives, including information assets criticality. Nevertheless, we used several ideas from this tool in this work, e.g. definition of access control matrix and security profile.

CORAS CORAS [LSS11] is a risk modeling method based on AS/NZS 4360:2004 standard [AS/04]. The language provides syntax and semantics to describe all phases of CORAS risk assessment: context establishment, risk identification, risk estimation, risk evaluation and treatment identification [Hea07]. Context establishment consists of identifying the stakeholders and their corresponding assets of the target of analysis, including their prioritization. During the risk identification phase security analysts construct threat diagrams, documenting how threat agents can exploit vulnerabilities in an application to cause unwanted incidents that harm assets. These diagrams give a high-level overview of the threat landscape and are well-suited for threat modeling. Risk estimation is performed based on the probabilities and impacts of unwanted incidents, normally using a predefined risk matrix. This identifies the subset of risks that are not tolerable and thus have to be further addressed. We use CORAS asset and risk diagrams in this work to provide a global easily comprehensible summary of risk assessment (see Sections 8.4.2 and 8.4.4).

Attack trees and attack patterns Attack trees construction is another threat modeling technique designed to represent in a structured way possible attacks on a system from the perspective of an attacker [Sch99]. The root of an attack tree is the ultimate goal of the attacker and is further refined through AND/OR decomposition into activities needed to achieve the goal [MEL01]. The leaf nodes usually denote system vulnerabilities, that an attacker could exploit to execute the intended attack scenario. Given additional behavioral information it is possible to automatically compute risks and derive the most probable attacks [Ing16].

Attack patterns are attack trees intended to describe knowledge of recurrent attacks. MITRE provides an extensive catalog of attack patterns with a classification taxonomy [CAP]. Microsoft use a catalog of threat tree patterns as a structured way to assess an architecture against existing threats during the STRIDE analysis [HL06, chapter 22]. A threat tree pattern has been proposed for each combination of DFD element type and applicable STRIDE threats. In contrast to simple checklists threat tree patterns are more structured and thus easier to follow and extend. Besides, the authors provide a list of design questions and attack applicability test criteria for each of the leaf nodes.

UML-based approaches Several approaches were introduced to address security concerns during the requirements phase of SDL. Among them are SecureUML, UMLSec, Abuse Cases and Misuse Cases [SO05].

SecureUML [LBD02] is a language and methodology for modeling authorization policies, in particular authorization constraints. It bases on a role-based access control (RBAC) model in combination with OCL constraints to capture application security requirements.

UMLSec [Jür02] is another UML-based language for modeling application security aspects. It provides means to specify and formally verify certain security requirements, e.g. fair exchange, confidentiality, secure information flow and others. Susceptibility to threats is modeled formally as a function that maps an adversary type and a component to possible (unwanted) operations. This work uses some of the concepts from UMLSec, such as different types of communication channels and associated threats. However, we chose a less formal approach which could be easier to follow for non-security experts.

Misuse Cases [SO05] and Abuse Cases are UML notations extending standard Use Cases to elicit security requirements. Their objective is to describe the undesirable behavior of a system, which contrasts with Use Cases, describing what it should do. The execution of Misuse Cases threaten specific Use Cases; and vice-versa some Use Cases can mitigate certain Misuse Cases. Misuse Cases also have informal textual descriptions that include threat scenario, attacker intention and capabilities (e.g. resources, skills and objectives).

Misuse Cases are similar to attack trees as the AND/OR node decomposition can be modeled with extend/include/generalization relationships. The CORAS notation used in this work encapsulates some concepts from Misuse Cases [Wei05].

Finally, UML activity and sequence diagrams can also be used along or instead of attack trees to describe possible attacks against an application [XN06].

Goal-based approaches Other approaches to modeling software security requirements are based on goal-oriented frameworks, such as NFR (**N**on-**F**unctional **R**equirements) framework or Tropos [MCY99]. Oladimeji et al. [OSC06] introduced an extension of the NFR framework, Softgoal Interdependency Graphs, for capturing threat-related concepts. NFR framework is specifically designed to model non-functional requirements, such as security. Softgoal Interdependency Graphs provide a high-level picture of security goals, threats, vulnerabilities, countermeasures and their interdependence. Moreover, they can be automatically analyzed for the satisfaction of the top goals.

Supakkul et al. [SHCO10] introduced goal-oriented security threat mitigation patterns to represent knowledge of common security problems and their solutions. They combined several notations, such as NFR framework, Problem Interdependency Graphs and Problem Frames to describe assets, security goals, undesirable outcomes, threats, vulnerabilities and mitigation solutions. The resulting diagrams are expressive and also amenable to automatic label analysis.

Formal approaches The discussion of the state of the art would be incomplete without mentioning several formal approaches to threat modeling, such as [DWTB03] and [XN06]. The main idea behind these techniques is to build the architectural and threat models using formal methods, such as Petri Nets and temporal logic. Then, model checking is used to verify if the design satisfies certain security properties. While they enable some advanced automatic reasoning, the main disadvantage is the need to fully specify the systems using rigorous notations. However, these activities are very costly [McG06, p. 211] and do not facilitate the communication with the customers.

Borozdin's extension of STRIDE Borozdin attempted to address some STRIDE shortcomings and extended it in the following directions [Bor11]:

1. STRIDE process was tailored to work with FMC/TAM block diagrams instead of original Data Flow Diagrams.

2. Architecture diagrams were enriched with additional semantics, such as component types and specific implementations; this information was used to refine STRIDE analysis, and query the vulnerability databases.

3. Question-based analysis allowed to guide the threat identification based on a predefined checklist of security questions.

4. Reachability analysis provided possible data flows on architecture diagrams.

He also developed a tool, which we used as a starting point to prototype the ideas in this work. We expanded over the work of Borozdin trying to address the following issues:

1. This framework does not allow specifying high-level assets and security objectives[1].

2. The access permissions are ambiguous: they are represented on FMC/TAM diagrams as informal CRUD labels, but it is not clear what storage they refer to.

3. There is no prioritization of identified threats and vulnerabilities: it is rarely possible and economically feasible to mitigate all identified risks.

4. Question-based analysis is quite laborious, requiring to ask a set of predefined leading questions for all architectural components.

8.3 Preliminaries

8.3.1 Terminology

The terms used in this chapter are mainly based on Common Criteria for Information Technology Security Evaluation [Com] and ISO 31000 [ISO].

Asset is an entity that the owner of the Target of Evaluation presumably places value upon (Common Criteria).

The term is misused in Microsoft Threat modeling literature and often means just data flow diagram element. Borozdin also calls an asset any FMC/TAM block diagram element, even application clients [Bor11, p. 31]. However, we argue that not every diagram element may be an asset and there may be assets, not present directly on the diagram.

Vulnerability is a weakness in the Target of Evaluation that can be used to violate the Security Functional Requirements in some environment (Common Criteria).

Threat is a potential attack, i.e. misuse or anomaly that violate the security goals (confidentiality, integrity, availability, etc.) of system's intended functions [XN06].

Threat agent or *threat source* is an entity that can adversely act on assets (Common Criteria).

Risk is the combination of the probability of an event and its consequence (ISO 31000)

Risk assessment is the overall process of risk identification, risk analysis and risk evaluation (ISO 31000).

[1]The meaning of the term *asset* used in this text is different from the one used by Borozdin. See Section 8.3.1

Risk analysis is the process to comprehend the nature of risk and to determine the level of risk (ISO 31000).

Threat model is a set of artifacts, containing the result of a threat analysis. This may include a description of application attack surface, list of vulnerabilities, threats in application environment and their corresponding risks, etc.

Threat analysis or *threat modeling* was introduced by Microsoft to denote, in fact, what is considered as risk assessment (see [McG06, p. 146] for a discussion on the term misuse). In this work threat analysis is used with the meaning originally proposed by Microsoft (i.e. essentially risk assessment) with the emphasis on identification of threats and vulnerabilities [HL06].

Stakeholder is a party, interested in successful operation of the system under analysis and possibly involved in the risk assessment process.

8.3.2 Case study

In this section we present the Flight Booking System (FBS) case study referenced throughout this chapter, an adaption of the case study from [Bor11]. It contains additional security-related aspects, such as trust zones and audit functionality as well as abstracts some irrelevant information. We decided to follow the FBS threat analysis case study to ensure generality and applicability of our results. By this ESCUDO-CLOUD contributes to automated threat analysis.

The FBS is owned by an airline and its main purpose is to sell flight tickets. Figure 8.1 shows the FBS Use Case diagram. The main actors are External Customer, Booking Agency, Ticket Seller, Cheap tickets website, Manager and Auditor. External Customer can order tickets via airline company web site. Ticket Seller works for the airline and can also place orders on behalf of the customers. Booking agency serves the same purpose, but is not part of the airline organization. Cheap tickets website provides users with an interface to view flights, but does not sell tickets itself. Manager maintains the prices and flight information, but also does not handle orders herself. Lastly, Auditor is responsible only for checking the audit logs to ensure compliance with company security policies.

Figure 8.2 presents the FBS FMC/TAM block diagram [TAMb]. The solution is a three-tier web application composed of presentation, business and persistence tiers: the presentation tier contains User Interface, Web Service and Log Admin Interface agents; the business tier is represented by Business Logic agent; and the persistence tier consists of Database and Logs storages. The most important data in the Database are: ticket orders; customers' private information or personally identifiable information; flight data (e.g. flight schedules, available places); and pricing policies.

8.4 Conceptual model

8.4.1 Overview

Figure 8.3 illustrates the Tam^2 high-level conceptual model and Figure 8.4 details it.

Asset model captures the security goals that our architecture must satisfy (assets and security objectives).

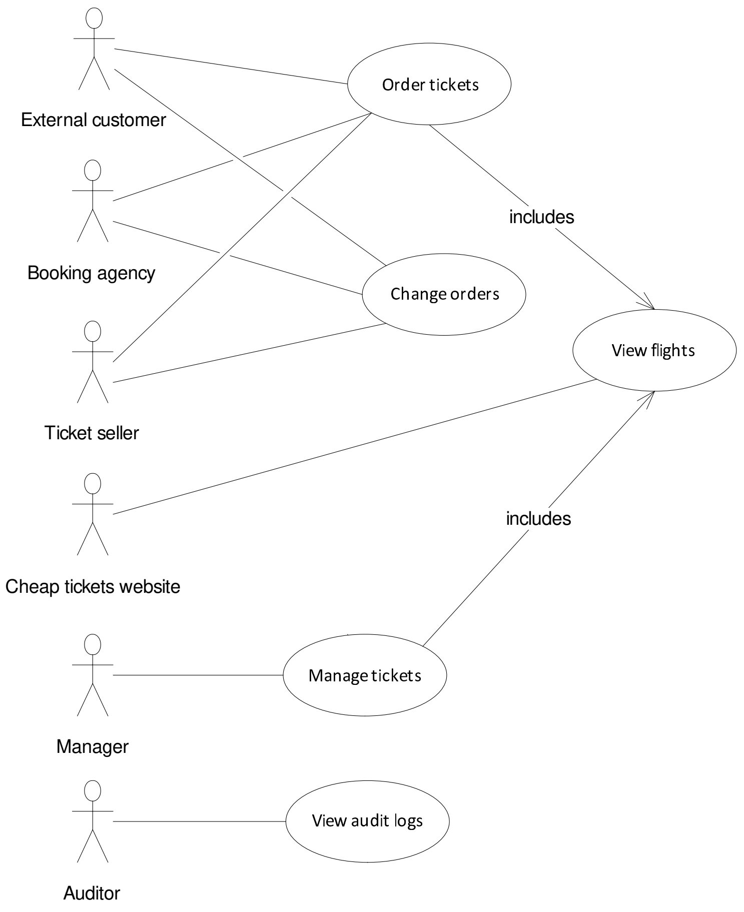


Figure 8.1: FBS use case diagram

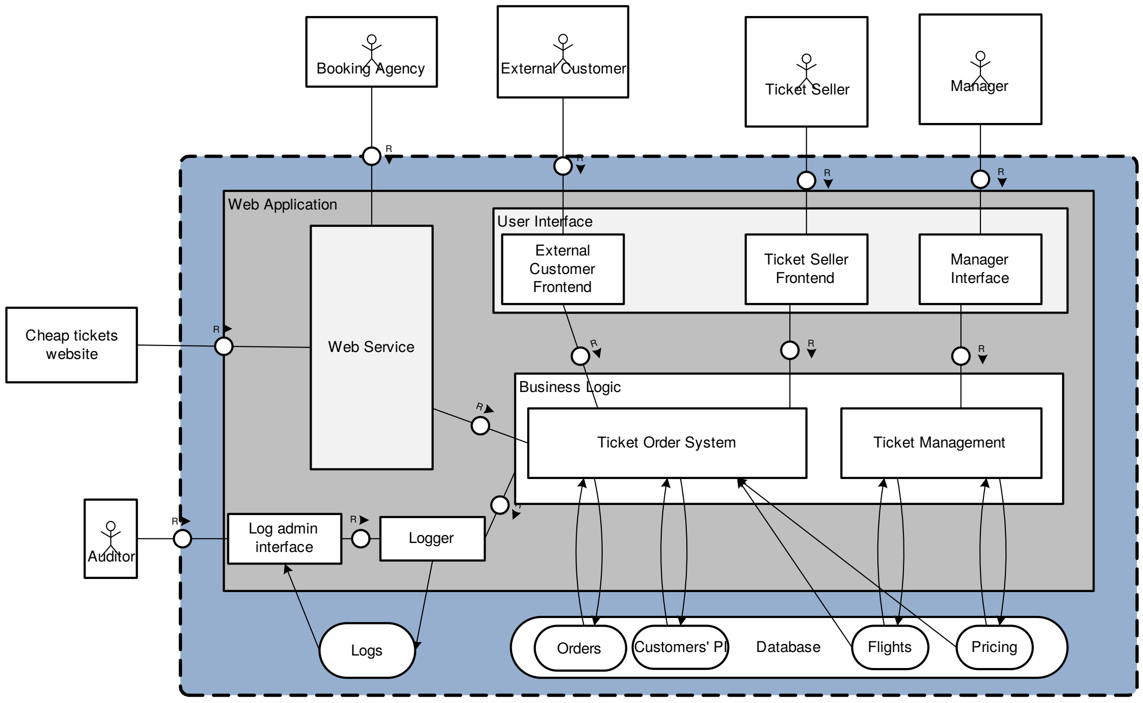


Figure 8.2: FBS TAM block diagram

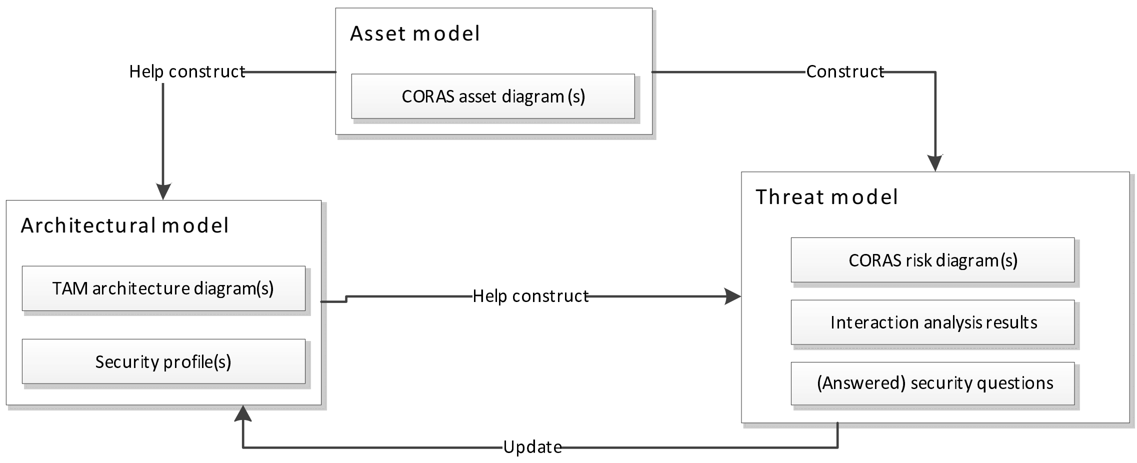


Figure 8.3: Conceptual model overview

Architectural model represents the high-level architectural design and security-related information (application architecture diagram and security profile).

Threat model summarizes the threat modeling results (risks, vulnerabilities, and threat agents).

This conceptual model provides means to extend the STRIDE software-centric threat modeling approach with the two other perspectives: asset-centric and attacker-centric. We used the CORAS notation to capture these two new perspectives and linked them to the architectural model. The following sections detail the presented conceptual model.

8.4.2 Asset model

Figure 8.8 shows the FBS CORAS asset diagram, consisting of stakeholders, assets and security objectives. Below we describe them in more detail.

Assets and stakeholders

Assets are the most valuable parts of the target of analysis (see Section 8.3.1). In contrast with software-centric approaches, such as STRIDE, we use an asset-centric approach, where primary assets guide the overall analysis. This permits to focus the analysis from the very beginning on the most important issues and as a result reduce its scope.

Assets do not exist alone but represent a value for some stakeholders. For example, in Figure 8.5 customers' personal information and flight data represent assets to the External Customer and Airline stakeholders.

Both stakeholders and tangible assets can be represented also on the architecture diagram: stakeholders take part in the system operation (e.g. External customer), and assets are agents or data storages containing some sensitive information (e.g. Customers' PI, Orders, Flights storages). Other assets, such as Company reputation and Conformance to regulations are intangible and hence cannot be present on the architecture diagram.

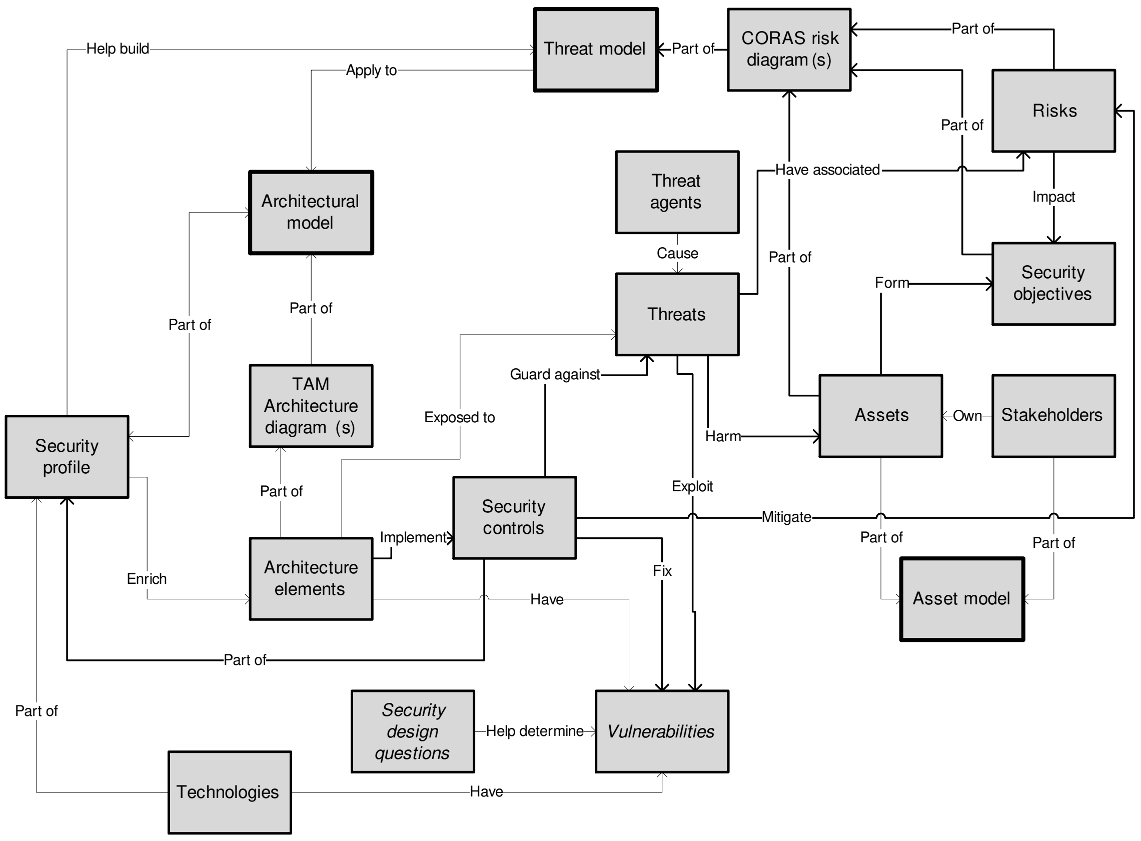


Figure 8.4: Detailed conceptual model

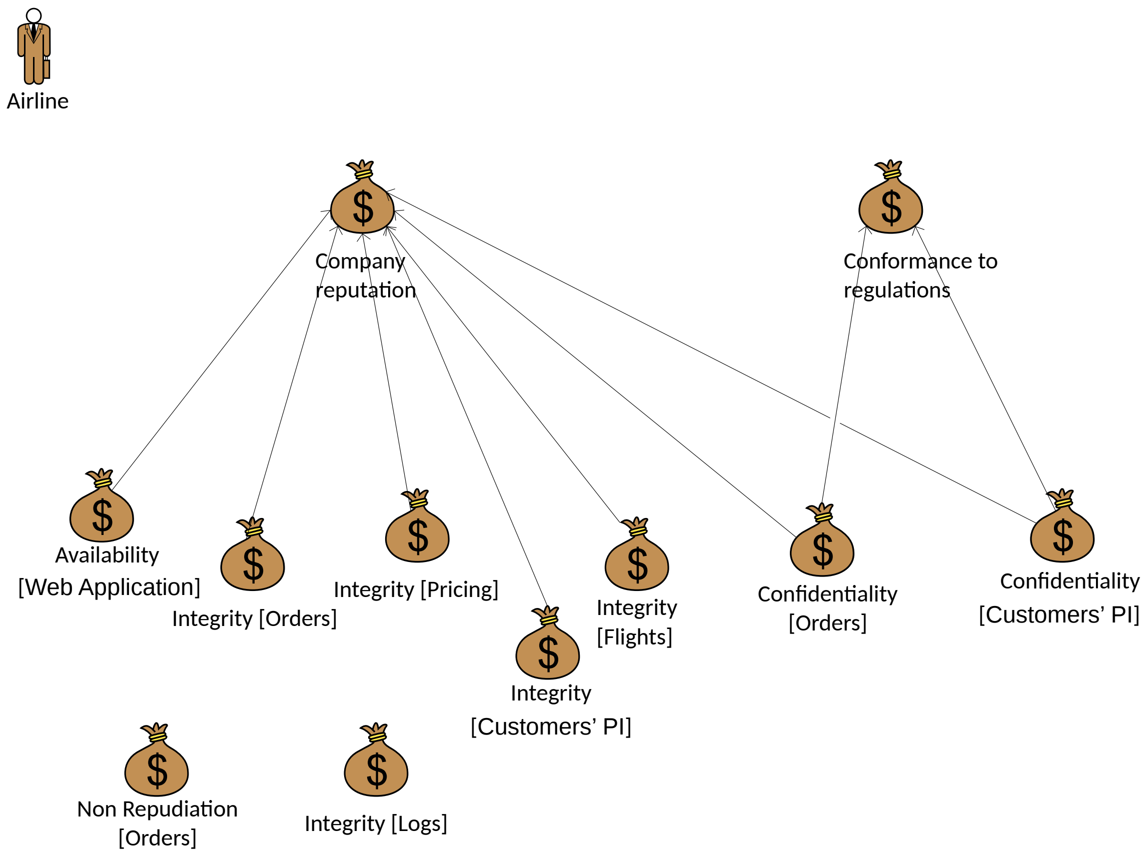


Figure 8.5: FBS Airline asset diagram

Security objectives

Assets may have several associated security properties required to preserve, e.g. confidentiality, integrity, availability and non-repudiation. *Security objective* is a required security property of a particular asset, e.g. Confidentiality [Orders] (see Figure 8.5).

We introduce this additional notion because not all security properties of an asset are usually important. For example, while Integrity [Flights] is a security objective, Confidentiality [Flights] is not, because of the public nature of this information. Secondly, not all security objectives are equally important to a stakeholder. For example, Confidentiality [Customers' PI] is more valuable than Integrity [Customers' PI] because, if violated, can hurt both company reputation and may lead to legal action. The relative priorities of security objectives are used to calculate the related risk levels (see Section 8.4.4). Finally, the security objectives instruct the interaction analysis on what kind of attack simulations to perform (see Section 8.5).

8.4.3 Architectural model

Figure 8.6 displays the underlying object model behind the Tam^2 tool, formalizing the architecture diagram elements and the security-related information. We have created a formal Alloy model defining these concepts.

Architecture diagrams

Architecture diagrams play an important role in identifying design-level threats and vulnerabilities in a system. Having a one-page overview of the whole application architecture — a tree forest-level view — can greatly facilitate the threat modeling process and avoid getting lost in details (see [McG06, p. 148]).

We use FMC/TAM [TAMb] UML-based notation for constructing architecture block diagrams, consisting of (human) agents, communication channels and data storages.

Agent is a process that performs operations on data and communicates with other agents (e.g. User Interface, Cheap tickets website, see Figure 8.2)

Human Agent is a user of the system (e.g. External Customer, Ticket Seller).

Storage is a passive element that holds the data accessed by agents (e.g. Logs, Orders).

Channel represents a request going from a sender to a receiver, that can be an Agent or a Human Agent. For example, a Ticket Seller can send a request to the Ticket Seller Front-end, which, in turn, can forward it to the Ticket Order System.

Access connects an agent to a storage and it can have different types (Read, Write, or Read/Write). For example, the Ticket Order System can read/write to Orders and Customers' PI data storages, however, it can only read from Flights and Pricing data storages.

A storage or an agent can be tangible assets. For example, Orders, Customers' PI, Flights, Pricing and Web Application are all assets, identified in the asset model. This link enables the interaction analysis to detect threats against sensitive diagram elements (see Section 8.5).

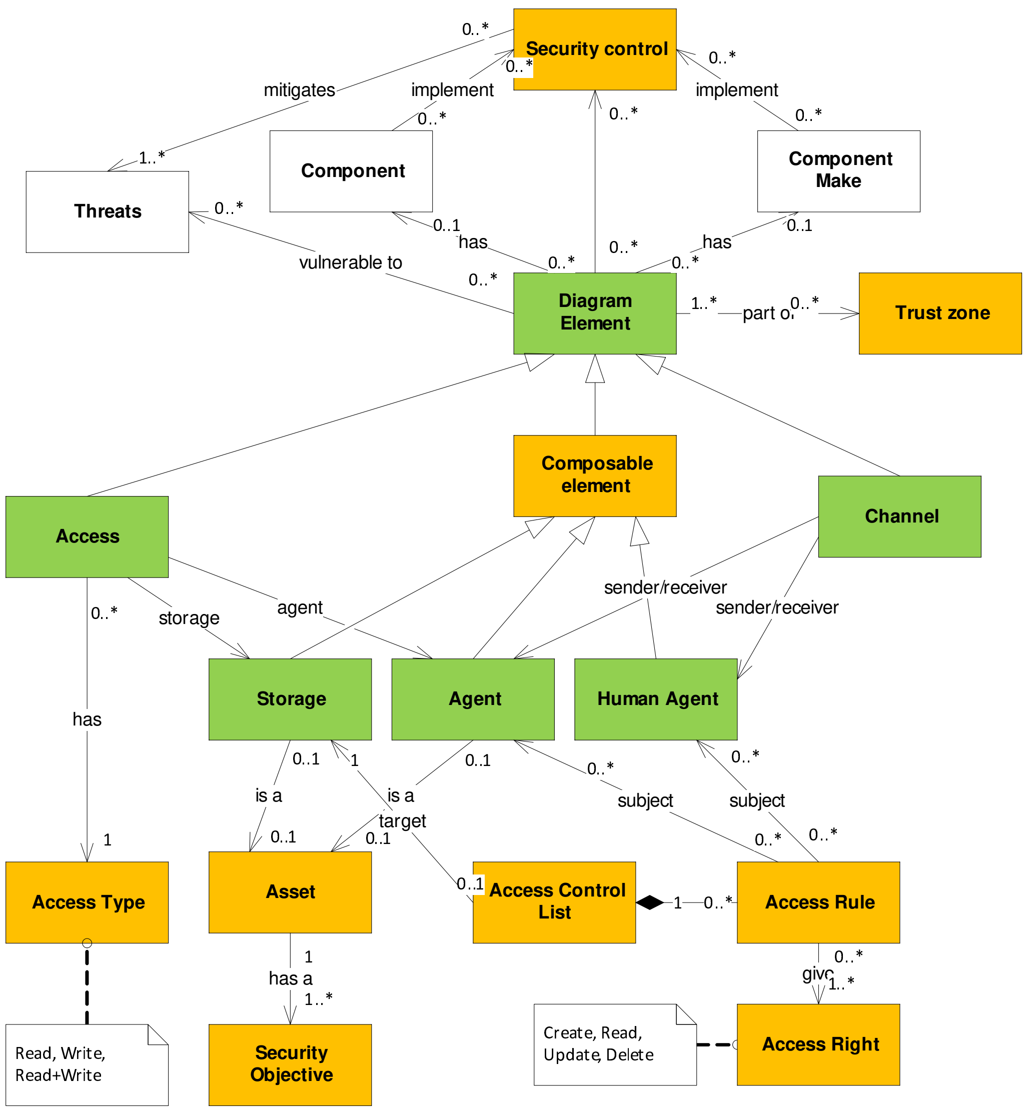


Figure 8.6: Architectural meta-model

Security profile

To facilitate the threat analysis it is essential to identify and document the application security-related aspects. Capturing this information explicitly reduces ambiguities and may enable automatic identification of the application threats and vulnerabilities. Unfortunately, the FMC/TAM standard does not provide means to specify security-related information, other than by informal text labels on the diagrams. We address this by capturing some security aspects, such as component types[2], trust zones, access control lists and security controls in the application security profile.

Trust zones Trust zones[3] are an important concept for providing an outside-in perspective of a system [McG06]. They indicate what components are internal (trusted) and external (untrusted or less trusted) to an application. Components separated by trust boundaries have different privilege levels or degrees of trust and thus require authentication and validation of the incoming data flows and possibly other security controls. We denote a trust zone as a dashed blue rectangle in the FMC/TAM block diagrams.

In our case study (see Figure 8.2) we have one trust zone, encompassing all web application, application logic and persistent layers. However, depending on the deployment model these can be three separate trust zones. Outside the trust zones remain the following agents: Auditor, Cheap tickets web site, Booking Agency, External Customer, Ticket Seller and Manager. This means that Web Application has to secure all entry points — Web Service, User Interface and Log Admin Interface — by implementing proper access control and data validation mechanisms, to avoid various malicious behavior by these agents.

Visualizing trust zones allows determining system entry points and thus focus the analysis on the most critical architectural components. In addition, the interaction analysis uses them to determine the threat sources and entry points for some possible attacks.

Access control lists The FMC/TAM standard allows specifying systems actors, but not their access permissions to the resources. However, this information is important for threat modeling. We postulate that even a simple access control model could enable detecting possible authorization problems at the conceptual design level. For this purpose we use access control lists (ACLs) to describe the access permissions (Create, Read, Update, Delete) the agents have on the storages (application resources).

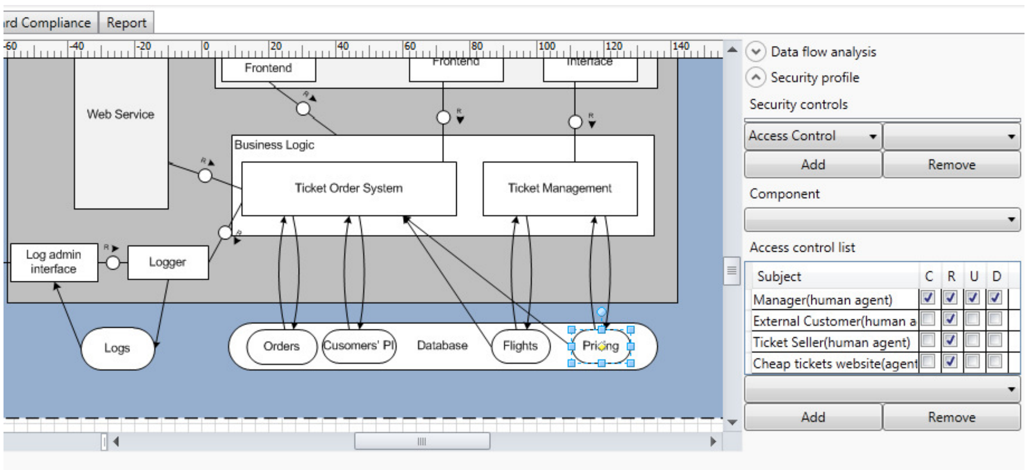
For example, a possible ACL for the Pricing storage is presented in Figure 8.7. It states that Manager has full control over the pricing data and the other agents can only read that data.

Besides documentation purposes, this information is used in the interaction analysis to determine how the data can propagate in the system. For instance, with appropriate access enforcement controls in place we can be sure that Ticket Seller cannot tamper with the pricing data.

Security controls Security controls or services are mechanisms mitigating some security threats to a system. Analyzing existing or planned controls in the architecture design is a key step in any

[2]*Component* and *Component Make* concepts were introduced by Borozdin [Bor11]: a component is the function of an architecture element, e.g. a Web Server, User Interface, Web Browser; a component make is a concrete implementation of a specific component, e.g. Internet Explorer 9.0, Microsoft SQL Server 2008. Their purpose is to link some functional semantics to the architecture diagram elements.

[3]Sometimes called trust domains, security domains or security realms



Access Control

Subject	C	R	U	D
Manager(human agent)	✓	✓	✓	✓
External Customer(human a		✓		
Ticket Seller(human agent)		✓		
Cheap tickets website(agent		✓		

Figure 8.7: ACL of a storage in Tam[2]

architectural threat modeling process. Based on the catalog from [NVDb] we propose a sample catalog of security controls grouped into categories: Authorization, Audit and Accountability, Identification and Authentication, Sensitive data Protection, Configuration management, Session management, System and Information integrity. A security control can be associated with components, component makes or architectural elements, in the decreased order of abstraction.

1. Authorization

   (a) Access Enforcement. The information system enforces assigned authorizations for controlling access to the system in accordance with applicable policy.

   (b) Information Flow Enforcement. The information system enforces assigned authorizations for controlling the flow of information within the system and between interconnected systems in accordance with applicable policy.

   (c) Unsuccessful Login Attempts. The information system enforces a limit of [Assignment: organization-defined number] consecutive invalid access attempts by a user during a [Assignment: organization-defined time period] time period.

   (d) Previous Logon Notification. The information system notifies the user, upon successful logon, of the date and time of the last logon, and the number of unsuccessful logon attempts since the last successful logon.

2. Audit and Accountability

   (a) Logging. The information system records events in a certain scope in order to provide an audit trail that can be used to understand the activity of the system and to diagnose problems.

   (b) Non-Repudiation of Origin. The information system can ensure that an entity that has performed an action cannot at a later time deny having done it.

3. Identification and Authentication

   (a) User Identification And Authentication. The information system uniquely identifies and authenticates users (or processes acting on behalf of users).

(b) Device Identification And Authentication. The information system identifies and authenticates specific devices before establishing a connection.

(c) Authenticator Feedback. The information system obscures feedback of authentication information during the authentication process to protect the information from possible exploitation/use by unauthorized individuals.

4. Sensitive data Protection

   (a) Denial Of Service Protection. The information system protects against or limits the effects of the following types of denial of service attacks: [Assignment: organization-defined list of types of denial of service attacks or reference to source for current list].
   (b) Transmission Integrity. The information system protects the integrity of transmitted information.
   (c) Transmission Confidentiality. The information system protects the confidentiality of transmitted information.
   (d) Network Disconnect. The information system terminates a network connection at the end of a session or after [Assignment: organization-defined time period] of inactivity.
   (e) Trusted Path. The information system establishes a trusted communications path between the user and the following security functions of the system: [Assignment: organization-defined security functions to include at a minimum, information system authentication and re-authentication].
   (f) Secure Name / Address Resolution Service (Authoritative Source). The information system that provides name/address resolution service provides additional data origin and integrity artifacts along with the authoritative data it returns in response to resolution queries.
   (g) Secure Name / Address Resolution Service (Recursive or Caching Resolver). The information system that provides name/address resolution service for local clients performs data origin authentication and data integrity verification on the resolution responses it receives from authoritative sources when requested by client systems.

5. Configuration management

   (a) Intrusion Detection. The information system monitors network or system activities for malicious activities or policy violations and produces reports to a Management Station.
   (b) Application Partitioning. The information system separates user functionality (including user interface services) from information system management functionality.

6. Session management

   (a) Concurrent Session Control. The information system limits the number of concurrent sessions for any user to [Assignment: organization-defined number of sessions].
   (b) Session Lock. The information system prevents further access to the system by initiating a session lock after [Assignment: organization-defined time period] of inactivity, and the session lock remains in effect until the user reestablishes access using appropriate identification and authentication procedures.

   (c) Session Termination. The information system automatically terminates a remote session after [Assignment: organization-defined time period] of inactivity.

   (d) Session Authenticity. The information system provides mechanisms to protect the authenticity of communications sessions.

7. Exception management

   (a) Error Handling. The information system identifies and handles error conditions in an expeditious manner without providing information that could be exploited by adversaries.

8. System and Information integrity

   (a) Input validation. Addresses: buffer overflow; cross-site scripting; SQL injection; canonicalization.

   (b) Spam Protection. The information system implements spam protection.

   (c) Malicious Code Protection. The information system implements malicious code protection.

   (d) Software And Information Integrity. The information system detects and protects against unauthorized changes to software and information.

In our case study, for example, we know that the communication channel between External Customer and External Customer Front-end is a HTTPS web channel. HTTPS has pre-associated security controls: Transmission Integrity and Transmission Confidentiality, which, in turn, alleviate confidentiality and integrity risks to the transmitted data.

For the External Customer Front-end, we know that this is a BC-FES-GUI 7.20 SAP GUI component. This particular component make is pre-associated with the following security controls: Access Enforcement, User Identification and Authentication and Input Validation. Security analysts may also decide to include on the diagram network-level security mechanisms (e.g. Firewalls, Intrusion Detection and Prevention Systems, DoS Defense System). The typical security controls these mechanisms implement are Denial of Service Protection, Information Flow Enforcement and Logging.

The main purpose behind specifying the security controls implemented by an application is to force security analysts to get more familiar with all security aspects of the system. This information can help in determining what threats are less likely in the system and thus focus on the ones that are not mitigated. Due to this layered structure, the security controls can be reusable across different architecture elements. In addition, the interaction analysis uses this data to restrict the possible data flows in the system.

8.4.4 Threat model

Figure 8.8 shows the FBS CORAS risk diagram, consisting of possible threat agents, risks and affected assets or security objectives. Below we describe them in more detail.

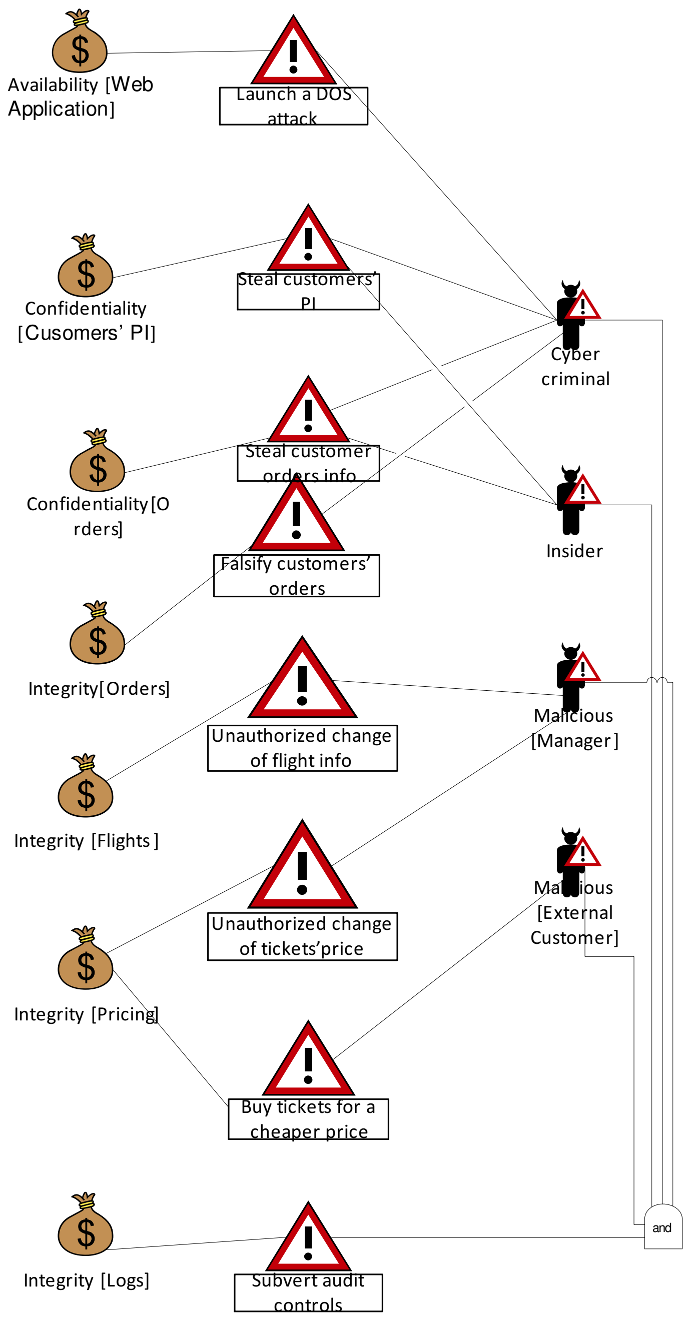

Availability [Web Application]
Launch a DOS attack
Confidentiality [Cusomers' PI]
Steal customers' PI
Confidentiality[Orders]
Steal customer orders info
Falsify customers' orders
Integrity[Orders]
Unauthorized change of flight info
Integrity [Flights]
Unauthorized change of tickets'price
Integrity [Pricing]
Buy tickets for a cheaper price
Integrity [Logs]
Subvert audit controls
Cyber criminal
Insider
Malicious [Manager]
Malicious [External Customer]
and

Figure 8.8: FBS risk diagram

Threat agents

Threat agents are malicious actors that can potentially compromise system assets or security objectives. Knowing who may attack a system is always beneficial, since then the security controls can be chosen proportionate to the threat agent's motivation and skills. This provides the attacker's perspective in the Tam^2 threat modeling process.

We differentiate between the following threat agent types[4]: motivated attacker, accidental discovery, script kiddies, organized crime, automated malware and the curious attacker; and origin: insiders and outsiders. Clearly, the security controls for insider and outsider threats will be completely different. In our case study we identified several possible threat agents (see Figure 8.8): both insiders (e.g. Malicious Manager) and outsiders (e.g. Malicious External Customer).

Some of the threat agents can be derived from architecture diagrams. Viewing all external actors (outside of the trust zone) as potentially malicious is helpful in identifying possible threats and entry points.

Risks

Risks in the threat model represent threats impacting certain assets or security objectives and the induced harm (risk level). They are characterized qualitatively by impact on assets and estimated frequency of occurrence. The risk levels are derived based on these two properties and the importance of the impacted assets[5].

Security questions

Borozdin introduced the question-based assessment of FMC/TAM block diagrams in [Bor11]. In his model the security questions belonged to STRIDE threat categories, which were associated with different diagram shapes, excluding meaningless combinations. Due to this coarse-grained association, given N diagram elements and M leading questions, the security analysts have to assess roughly $N * M$ possible combinations, even though many of them do not make sense.

Here we show how to reduce the laboriousness of his approach by taking advantage of the enriched semantics from the architectural model. Figure 8.9 provides the conceptual model on relationships between security questions and other concepts (the bold lines indicate new associations and the colored boxes — new concepts).

We noticed that some questions from a threat category are meaningful only for specific diagram shapes. For example, *“An attacker can spoof a server because identifiers aren't stored on the client and checked for consistency on re-connection”* applies only to agents; while *"An attacker can write to a data store your code relies on"* — applies only to storages.

Further, many questions are meaningful only for specific security controls. For example, *“An attacker could try one credential after another and there's nothing to slow them down”* applies only to diagram elements implementing authentication mechanisms (e.g. User Interface, Log admin interface, and Web Service agents); *“An attacker can enter data that is checked while still under their control and used later on the other side of a trust boundary”* — applies to input validation; and *"An attacker can make the logs wrap around and lose data"* — applies to logging (the Logger agent in FBS).

[4]Other various indicators can also be used to describe threat capabilities.

[5]We use a simple formula for calculating the risk level as Impact * Frequency * Importance. However, other risk evaluation techniques, such as CVSS, may be more appropriate for real-life scenarios.

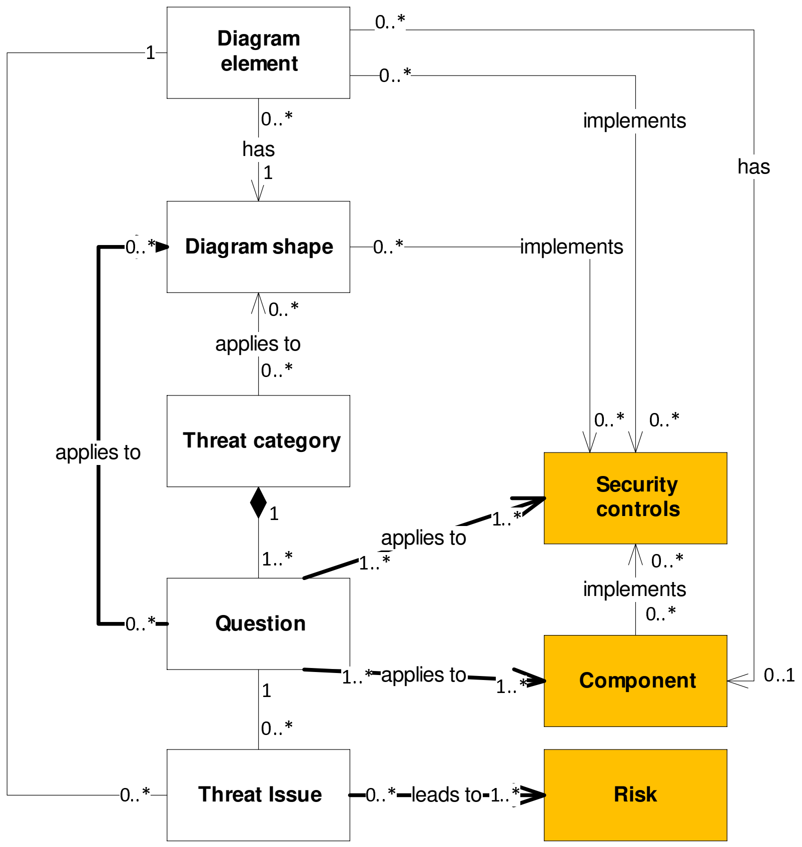
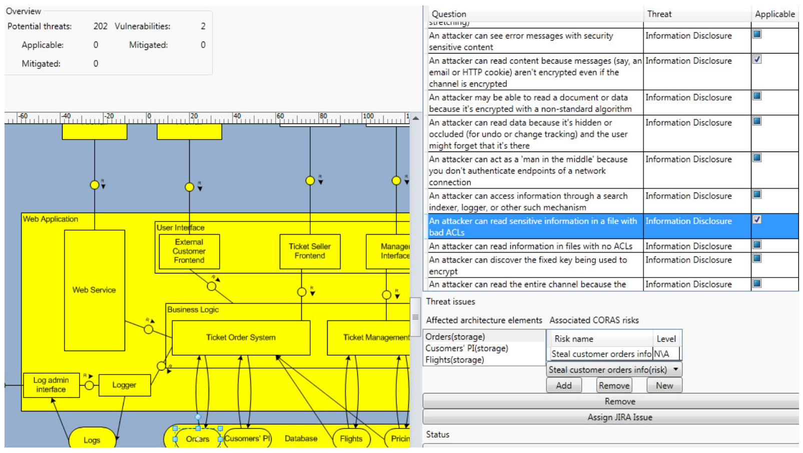


Figure 8.9: Questions association model



Overview

Potential threats:	202	Vulnerabilities:	2
Applicable:	0	Mitigated:	0
Mitigated:	0		

Question	Threat	Applicable
stretching)		
An attacker can see error messages with security sensitive content	Information Disclosure	☐
An attacker can read content because messages (say, an email or HTTP cookie) aren't encrypted even if the channel is encrypted	Information Disclosure	✓
An attacker may be able to read a document or data because it's encrypted with a non-standard algorithm	Information Disclosure	☐
An attacker can read data because it's hidden or occluded (for undo or change tracking) and the user might forget that it's there	Information Disclosure	☐
An attacker can act as a 'man in the middle' because you don't authenticate endpoints of a network connection	Information Disclosure	☐
An attacker can access information through a search indexer, logger, or other such mechanism	Information Disclosure	☐
An attacker can read sensitive information in a file with bad ACLs	Information Disclosure	✓
An attacker can read information in files with no ACLs	Information Disclosure	☐
An attacker can discover the fixed key being used to encrypt	Information Disclosure	☐
An attacker can read the entire channel because the	Information Disclosure	☐

Threat issues

Affected architecture elements: Orders(storage), Cusomers' PI(storage), Flights(storage)

Associated CORAS risks

Risk name	Level
Steal customer orders info	N\A

Steal customer orders info(risk)

Figure 8.10: Question-based analysis in Tam[2]

Finally, some questions can be attributed only to specific component types. For example, *"An attacker can make a server unavailable or unusable but the problem goes away when the attacker stops"* applies only to diagram elements with the Web Server component (Web Application agent in FBS).

This way we bring context awareness in the question-based assessment and as a result reduce the scope of threat identification.

A new association has been added to link the identified threat issues to high-level risks in the CORAS risk diagram. For example, if the question *"An attacker can read sensitive information in a file with bad ACLs"*[6] is applicable to the Orders storage (see Figure 8.10), then the associated risk is "Steal customer orders info". This provides a rationale on why fixing a particular threat issue is important and helps to determine the priority of corresponding issue tickets.

8.5 Interaction analysis

Interaction analysis is an asset-centric threat identification approach that automatically finds some threats to security objectives (from the asset model) based on architecture specification (from the architectural model). In particular, it takes into account component interactions on the architecture diagrams[7], parent-child relationships, component types, trust zones, present security controls and access permissions. For this reason, the analysis has to be preceded by asset identification and architecture specification steps (Sections 8.6.2 and 8.6.3).

Interaction analysis checks a predefined set of threats (malicious input, impersonation, unauthorized access, data integrity or origin falsification) to the following supported security properties: confidentiality, integrity and accountability. While the recognized threats constitute a small subset of all potential threats, they are among the most popular. As a result, interaction analysis identifies the necessity for the following security controls: input validation, authentication, authorization, logging, non-repudiation of origin.

Some of the identified issues will be due to an incomplete architecture specification and others due to design errors. In the prior case the analysts need to fix the specification, and in the later they can create corresponding tickets in the issue tracker.

[6]In this case the file is actually a storage

[7]This is the reason for the name.

The following sections detail the functioning of interaction analysis, supported by examples of identified threats.

8.5.1 Data flows

In this section we present an abstraction of the requests inside a system, possible threats, and an algorithm for simulating the data flow propagation.

Listing 8.1 shows the definition of the *Request* type in Alloy language[8] [Jac12]. A request can be initiated by an agent or human agent, and contains some operation on the target storage. To imitate several types of attacks we use sticky properties indicating whether the request is spoofed and/or otherwise malicious.

```
abstract sig Request {
  subject: Agent + HumanAgent,
  operation: AccessType,
  target: Storage,
// malicious properties
  falsifiedOrigin: Bool,
  falsifiedData: Bool,
  maliciousInput: Bool,
  dos: Bool
}
```

Listing 8.1: Request type definition

Algorithm 1 shows how to compute all reachable diagram elements given a request[9]. For that we return all the diagram elements starting from request subject, following the attached channels and accesses. In addition to this, (a) agents implementing certain security controls can filter out some bad requests (see Listing 8.2); (b) we consider the parent-child relationships between diagram elements, following the FMC/TAM standard.

Listing 8.2 shows how the requests are validated by the security controls implemented by agents. These types of security controls are taken into account at the moment[10]: *User Identification And Authentication*, *Access Enforcement* (*Authorization*), *Input Validation*, *Non Repudiation of Origin* and *Denial Of Service Protection*. For example, the *AuthorizationControl* checks whether the operation in the request is allowed in the ACL of the target storage.

Below we show how this simple model can be used to detect some possible violations of confidentiality, integrity and accountability security properties.

8.5.2 Threat identification

Confidentiality

Listing 8.3 shows how Tam[2] checks for some possible confidentiality threats to an asset. For that it simulates several types of requests from all the external agents (human or non), and checks

[8]In the beginning we used Alloy to describe the architectural model and implement the interaction analysis. It proved the concept, but since the model checking was slow, we implemented it algorithmically in Tam[2].

[9]'*' denotes the transitive closure of a relation.

[10]Other security controls from the catalog (see Section 8.4.3) could also be used to check against some threats. For example, *Unsuccessful Login Attempts* — against brute-forcing the authentication, or *Session management* — against session hijacking.

Algorithm 1 Data flows propagation

```
function REACHABLE($request$)
    for all $elem$ in $HumanAgents$ do
        for all $channel$ in $elem.OutgoingChannels$ do
            $elem.next \leftarrow elem.next \cup channel.Receiver$
        end for
    end for
    for all $elem$ in $Agents$ do
        if ALLOWEDBYSECURITYCONTROLS($elem$,$request$) then
            for all $channel$ in $elem.OutgoingChannels$ do
                $elem.next \leftarrow elem.next \cup channel.Receiver$
            end for
            for all $access$ in $elem.Accesses$ do
                if $access.Type = request.Operation$ then
                    $elem.next \leftarrow elem.next \cup channel.Receiver$
                end if
            end for
            $elem.next \leftarrow elem.next \cup elem.parent.next$
        end if
    end for
    for all $elem$ in $Agents \cup HumanAgents$ do
        $elem.next \leftarrow elem.next \cup elem.next.*child$
    end for
    return $request.subject.*next$
end function
```

```
pred AllowedBySecurityControls [a: Agent, r:Request] {
  // authorization security control
  (no AuthorizationControl & a.allSecurityControls ||
    some AuthorizationControl & a.allSecurityControls && AccessAllow[r]) &&
  // authentication security control
  (no AuthenticationControl & a.allSecurityControls ||
    some AuthenticationControl & a.allSecurityControls && Original[r]) &&
  // non−repudiation security control
  (no NonRepudiationControl & a.allSecurityControls ||
    some NonRepudiationControl &
      a.allSecurityControls && Original[r] && ValidData[r]) &&
  // input validation security control
  (no InputValidationControl & a.allSecurityControls ||
    some InputValidationControl & a.allSecurityControls &&
      ValidInput[request]) &&
  // DOS protection security control
  (no DOSProtection & a.allSecurityControls ||
    some DOSProtection & a.allSecurityControls && isFalse[request.dos])
}

pred AccessAllow [request: Request] {
  let rights = request.target.acl[request.subject] |
  request.operation in rights
}

pred Original [request: Request] {
  isFalse[request.falsifiedOrigin]
}

pred ValidInput [request: Request] {
  isFalse[request.maliciousInput]
}

pref ValidData [request: Request] {
  isFalse[request.falsifiedData]
}
```

Listing 8.2: Request filtering by security controls

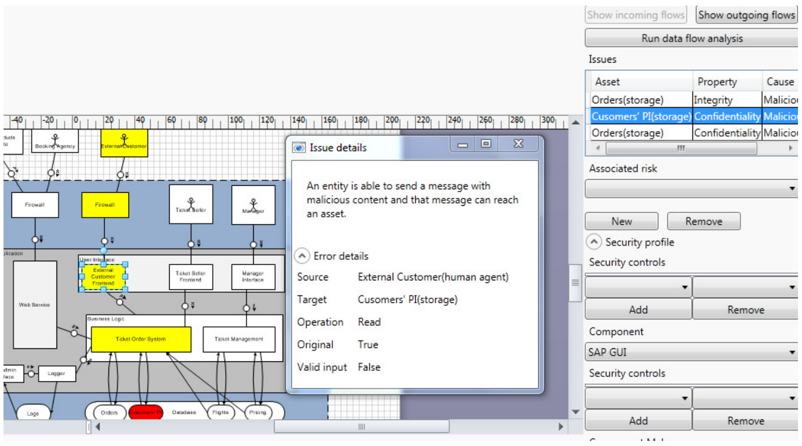
that an agent can read information from an asset (storage) only if the request has been previously validated (i.e. it is well formed, authenticated and authorized), and the information is protected in transit outside of the trust zone (e.g. the channels are encrypted).

Figure 8.11 shows an example of a possible confidentiality issue. In this case an External Customer is able to send a *read* request with malformed contents and that request is never filtered out on the way to the Customers' PI storage. This can potentially cause leakage of private information, e.g. by crafting an SQL statement that returns all the contents of the storage, bypassing input validation. The Tam^2 tool describes the issue details and illustrates the malicious flows on the architecture diagram.

To fix this particular issue the security analysts have to check if e.g. External Customer Front-end implements proper validation of input requests, and, if so, specify this in the security profile.

```
// only authorized requests are allowed to read AND
// information is protected in transit
pred CheckConfidentiality[asset:Asset]{
  all a: HumanAgent + Agent − TrustBoundary.contents|
    (all rf:RequestFlow, r:rf.request |
    r.subject = a && r.target = asset &&
    r.operation in AccessRead && asset in rf.reachable
      => Original[r] && AccessAllow[r] &&
        (all a1: Element − TrustBoundary.contents |
          a1 in rf.reachable − r.source =>
          TransmissionConfidentialityControl in a1.allSecurityControls))
}
```

Listing 8.3: Interaction analysis: confidentiality


Issues

Asset	Property	Cause
Orders(storage)	Integrity	Malicio
Cusomers' PI(storage)	Confidentiality	Malicio
Orders(storage)	Confidentiality	Malicio

Issue details

An entity is able to send a message with malicious content and that message can reach an asset.

Error details

Source	External Customer(human agent)
Target	Cusomers' PI(storage)
Operation	Read
Original	True
Valid input	False

Component: SAP GUI

Figure 8.11: Confidentiality issue properties in Tam^2

Integrity

Similarly, listing 8.4 shows how Tam^2 checks for some possible integrity threats to an asset.

Figure 8.12 shows an example of a possible integrity issue. In this case someone impersonating a Ticket Seller is able to send a *write* request and that request can reach the Orders storage. As ticket sellers have the write privileges to the Orders storage, this can potentially lead to someone tampering with the orders data.

To fix this issue the security analysts have to check if e.g. Ticket Seller Front-end implements proper authentication mechanisms, and, if so, specify this in the security profile.

Accountability

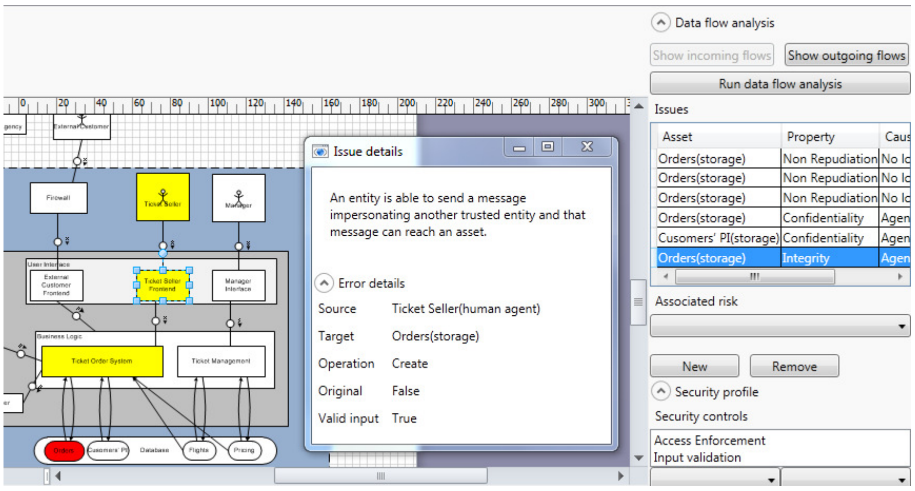
Another property a secure design should ideally have is accountability. In our interpretation this means that all requests to system assets (a) are logged; and (b) are original and the data origin and integrity are verified. Listing 8.5 shows how Tam^2 checks for some possible accountability threats.

In FBS there is a logging agent[11], so that the requests through Business Logic should be

[11] At the conceptual design level we are not concerned with OS/file system-level logging.

```
// only authorized requests are allowed to write AND
// information is protected in transit
pred CheckIntegrity[asset:Asset]{
  all a: HumanAgent + Agent − TrustBoundary.contents|
    (all rf:RequestFlow, r:rf.request |
    r.subject = a && r.target = asset &&
    r.operation in AccessWrite && asset in rf.reachable
      => Original[r] && AccessAllow[r] && ValidInput[r] &&
        (all a1: Element − TrustBoundary.contents |
          a1 in rf.reachable − r.source =>
          TransmissionIntegrityControl in a1.allSecurityControls))
}
```

Listing 8.4: Interaction analysis: integrity



Issue details

An entity is able to send a message impersonating another trusted entity and that message can reach an asset.

Error details

Source	Ticket Seller(human agent)
Target	Orders(storage)
Operation	Create
Original	False
Valid input	True

Issues

Asset	Property	Caus
Orders(storage)	Non Repudiation	No lo
Orders(storage)	Non Repudiation	No lo
Orders(storage)	Non Repudiation	No lo
Orders(storage)	Confidentiality	Agen
Cusomers' PI(storage)	Confidentiality	Agen
Orders(storage)	Integrity	Agen

Security controls

Access Enforcement
Input validation

Figure 8.12: Integrity violation flow in Tam^2

logged[12]. The audit logs are kept in the Logs storage with very limited access permissions (read-only for Auditor).

In addition, to ensure non-repudiation the External Customer, Ticket Seller and Manager Front-end implement the Non-Repudiation of Origin security control (e.g. using digital signatures).

Defense in depth

Secure systems rely on the defense in depth principle so that if one security control fails the system assets are not compromised. A useful exercise would be to deliberately turn off the security controls one by one and see what kind of threats this can lead to. If some valuable assets are compromised, following a cost-benefit analysis, security architects may decide to put in place additional security controls for backup.

In FBS, for instance, it may happen that an attacker bypasses the Input Validation control and, as a result, compromises the machine, hosting the User Interface (i.e. gets inside the trust zone). If sensitive data (e.g. Customers' PI) travels unprotected the attacker could gain access to

[12]Of course, having only the architectural design it is impossible to say whether the Business Logic indeed logs all important events. This should be ensured during the implementation and testing phases of the SDL.

```
pred CheckAccountability[asset:Asset]{
  CheckLogging[asset] && CheckNonRepudiation[asset]
}
// all requests to the assets have to be logged
pred CheckLogging[asset:Asset]{
  all a: HumanAgent + Agent − TrustBoundary.contents|
    (all rf:RequestFlow, r:rf.request |
    r.subject = a && r.target = asset =>
    (some a1: Agent | a1 in rf.reachable &&
      LoggerControl in a1.allSecurityControls))
}
// only original and integral requests can reach their target
pred CheckNonRepudiation[asset:Asset]{
  all a: HumanAgent + Agent − TrustBoundary.contents|
    (all rf:RequestFlow, r:rf.request |
    r.subject = a && r.target = asset && asset in rf.reachable =>
        Original[r] && ValidData[r])
}
```

Listing 8.5: Interaction analysis: accountability

it. In case the risk of this event is unacceptable the security architects may decide to encrypt the data between User Interface and Business Logic (i.e. implement the Transmission Confidentiality security control). The same is true for storage: it may be necessary to encrypt the Customers' PI in case someone gets unauthorized access to the machine with the database.

8.6 Threat modeling process

This section presents the Tam^2 architectural threat modeling process based on the above conceptual model.

8.6.1 Overview

Section 8.2 provided an overview of several threat modeling methods, emphasizing STRIDE. Figure 8.13 illustrates a reference architectural threat modeling process used throughout this work[13].

The process addresses the architectural design phase of SDL and can be thus preceded by other threat modeling techniques from the requirements analysis phase, e.g. Misuse Cases (see Section 8.2). In such case we would have as an input from the requirements phase, besides a list of functional requirements, also initial security objectives and optionally some security requirements.

The aim of the architectural threat modeling process is to evaluate an architectural design against possible threats and derive a list of risks that have to be addressed. On each iteration the interested parties (e.g. security analysts and software architects) identify application security objectives, create (update) the architecture and evaluate the design against risks.

Asset identification The process starts with identifying stakeholders, assets and security objectives of the application as part of the asset model (see Sections 8.4.2 and 8.6.2).

[13]In essence it is similar to STRIDE and other risk assessment processes, but highlights the elements relevant to our discussion.

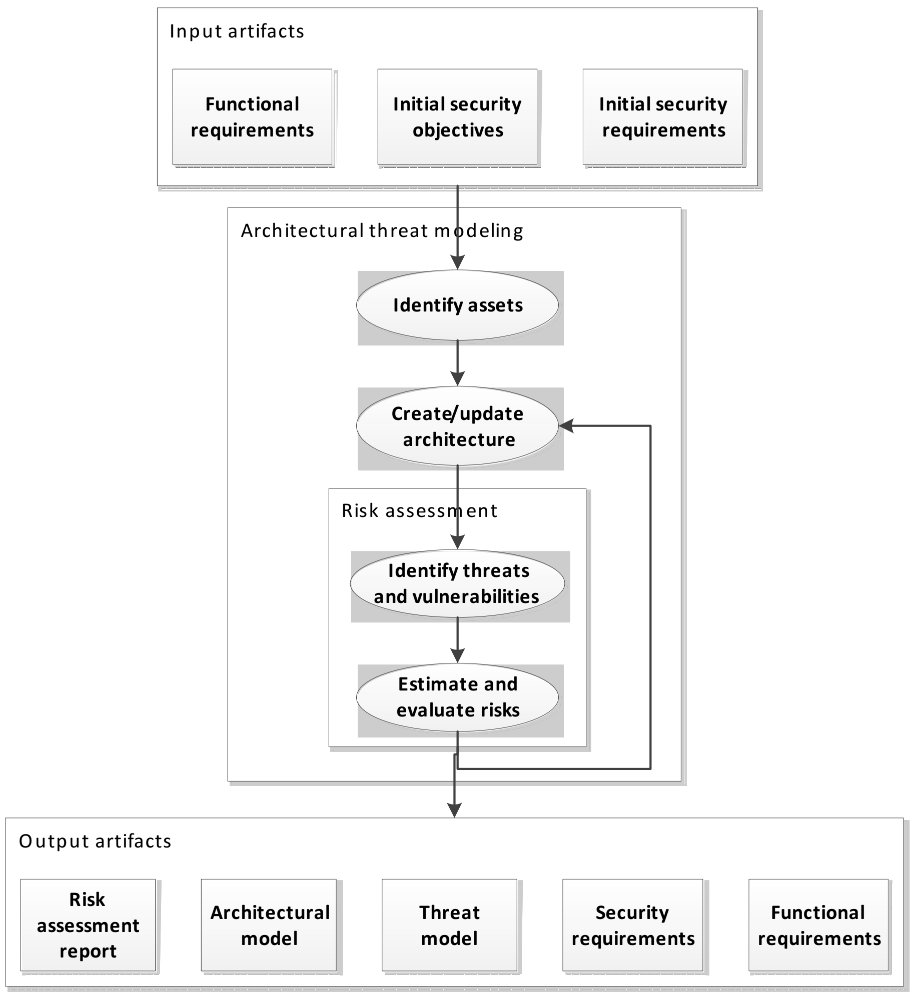


Figure 8.13: Architectural threat modeling process

Architecture specification The next step is designing the application based on the specified functional and security requirements. The produced artifacts constitute the architectural model (see Sections 8.4.3 and 8.6.3).

Risk assessment The core of the threat analysis is evaluating the architectural design against possible threats and determining their risks. We provide means to automate identification of threats and vulnerabilities in the context of the specified security objectives and architectural model. This is followed by risk estimation and evaluation. The results of the risk assessment make up the threat model (see Sections 8.4.4 and 8.6.4).

The main results of the threat modeling process are a risk assessment report, an (updated) architectural model and a threat model. Besides, new security requirements can emerge and even functional requirements can change[14].

After each iteration, if unacceptable risks were found, new security requirements are created to address them and the architectural model is updated with corresponding security controls. If the threat analysis yields only acceptable risks the process ends. It is important to rerun the threat analysis each time the architecture changes and regularly (e.g. once a year) to keep pace with the evolving threat landscape.

In some cases other sequences of these steps are more appropriate. For example, one can analyze threats before having an architectural design because they are inherent to the application operation environment. On the other hand, to analyze the vulnerabilities of an application it is necessary to have an architecture first.

In the following sections we illustrate each step of the Tam^2 process on FBS.

8.6.2 Asset identification

We suggest constructing the asset model (see Section 8.4.2) in the following sequence:

1. Specify the stakeholders: Airline, External Customer and Booking Agency.

2. Specify the top-level assets for each stakeholder: Company reputation and Conformance to regulations.

3. Specify sensitive data in the system: Orders, Pricing data, Flight data and Customers' PI.

4. Specify security properties of primary interest, e.g. confidentiality, integrity and non-repudiation of Orders.

5. Set the relative priorities of the assets and security objectives.

8.6.3 Architecture specification

In this section we show how to construct the architectural model (see Section 8.4.3).

[14]Howard provided an interesting example on how the Microsoft developers had to exclude a whole feature from Windows Vista because the threat analysis showed that it is vulnerable to a number of threats that cannot be easily mitigated [Thr]

Architecture diagram construction

To get optimum results from the Tam^2 risk assessment phase (see Section 8.6.4), we suggest the following sequence for creating FMC/TAM architecture diagrams (see Section 8.4.3):

1. Specify all application high-level components: agents, data storages and channels. In particular, represent all tangible assets from the asset model.
2. Specify all external agents (both human and non-human) and all related entry points.
3. Represent other security-relevant functionality, such as administration, auditing, etc[15]. For instance, in FBS we explicitly pointed the auditing and logging components: Log admin interface, Logger and Log storage.

The diagrams should be kept at the conceptual design level and not cluttered with implementation details. Removing irrelevant components and grouping related entities reduce the number of diagram elements and hence simplify the threat analysis. For example, in FBS we don't consider about the internals of the Cheap tickers website as it is outside of the application boundaries.

Security profile specification

To specify the Tam^2 application security profile (see Section 8.4.3):

1. Specify all known component and component makes for architecture diagram elements.
2. Outline all trust zones: groups of components having the same privilege level (degree of trust).
3. Specify access permissions for all the storages in the architecture diagrams (see Figure 8.7 for an example ACL).
4. Specify implemented security controls for architecture diagram elements (if not already pre-associated with the related component and component make).

See Section 8.4.3 for sample usage of components, component makes and security controls in FBS.

8.6.4 Risk assessment

The Tam^2 risk assessment consists of (a) threat and vulnerability[16] identification, (b) risk estimation and evaluation (see Figure 8.13). The threat identification is performed by interaction (Section 8.6.4) and question-based (Section 8.6.4) analyses. They are complementary, since the interaction analysis takes a top-down (asset-centric) approach and question-based analysis uses a bottom-up (software-centric) approach. Risk estimation and evaluation is briefly described in Section 8.6.4.

[15]In real life setting the high-level architecture diagrams are likely to lack security-related features; however, they are crucial for the threat analysis.

[16]For vulnerability identification we kept the Borozdin's approach [Bor11]. Borozdin extracts the vulnerabilities from the open vulnerability databases based on the component makes of architecture diagram elements.

Interaction analysis

To perform interaction analysis the user follows the following steps:

1. Run the interaction analysis algorithm and examine each of the identified issues.
2. Eliminate the false positives by updating the security profile with relevant information (e.g. add missing security controls).
3. For the remaining issues specify the associated risk from the CORAS risk diagram[17].
4. If the corresponding risk is absent, create new one[18].

Question-based analysis

Section 8.4.4 describes the conceptual model and the rationale behind the question-based analysis. The process is basically the same as in [Bor11], but enhanced with the refinement of the security questions based on the information from the security profile[19].

Risk estimation and evaluation

Once the CORAS risk diagram is ready, the risks can be processed in the following sequence:

1. Analyze the risks and specify the associated qualitative attributes: Frequency and Impact.
2. Risk levels will be automatically derived from these values and the importance of the impacted security objective[20].
3. Based on the risk levels, decide on the risk mitigation strategy: acceptance, transfer or treatment.
4. Create issue tickets for unacceptable issues with the priorities proportionate to the risk levels[21].

8.7 Results

This section evaluates the proposed threat modeling process and presents its limitations and possible applications.

8.7.1 Evaluation

We will try to speculate here on the efficiency of this solution comparing it to STRIDE, considering the following key performance indicators:

- Time of risk assessment

[17]The Tam^2 tool will propose a list of risks impacting the corresponding security objectives.

[18]The Tam^2 tool will automatically create the corresponding threat agent, risk and the associations on the CORAS risk diagram based on the information from the identified issue.

[19]The Tam^2 tool automatically highlights the potentially vulnerable diagram elements. Besides, the identified threat issues can be later linked with high-level risks from the CORAS risk diagram.

[20]Tam^2 uses a scale of five risk levels: Insignificant, Minor, Medium, Major and Catastrophic.

[21]The Tam^2 tool is integrated with JIRA issue tracker.

- Precision of risk estimation and prioritization
- Coverage of threats and vulnerabilities

The time required for Tam^2 risk assessment should be less than for STRIDE since the information specified in the architectural model decreases the laboriousness of the question-based assessment by filtering out the irrelevant questions. This should be noticeable for big diagrams, because for STRIDE the time complexity is $O(NE * NQ)$, where NE is the number of diagram elements and NQ is the number of leading questions. Of course, the security analysts will need to spend some time to create the FMC/TAM block diagrams and specify the application security profile. However, we speculate that studying the design documents (which security analysts will have to do anyway) usually takes much more time than capturing this summary in Tam^2. Additionally, Tam^2 may automatically query the vulnerability databases using the information from the asset model, so vulnerability identification should also take less time.

As for precision, Tam^2 initially focuses on system assets; the threats identified by interaction analysis usually are easier to rank, since we can see how they impact the assets. In contrast, STRIDE uses a software-centric approach, so the criticality of the identified threats may not be as obvious.

Finally, the coverage of threats and vulnerabilities in STRIDE should not differ from Tam^2.

8.7.2 Limitations

Below we describe several limitations on the usability of this work:

- We found it difficult to come up with a good taxonomy for security-related information (e.g. component types and security controls). The challenge here is that the classification has to be useful across many types of applications.
- The interaction analysis finds only a relatively small number of possible issues. An advanced reasoning would require a more complex architectural model based on formal methods. But this would also considerably increase the complexity of the threat modeling and hence fewer people would be able to master it.
- Having just a one-page block diagram limits the amount of design information possible to express. To get a more complete view on the application design it needs to be refined into other block or activity/state diagrams.
- The information in the security profile is quite limited. Ideally, it should capture as much security-related aspects as possible to provide a comprehensive view on the application security state of affairs. For instance, in addition to specifying that a component performs authentication, one could detail what mechanisms are used for that: certificates, passwords, etc; or in addition to saying that a storage implements sensitive data protection, one could detail what type of encryption is used and how the encryption keys are secured. Also, a more fine-grained access control model (e.g. RBAC) may be a better choice instead of the current ACL-based.
- Tam^2 can only capture a small portion of the information needed for risk assessment. Many more relevant artifacts (e.g. detailed architecture specifications, cost-benefit analysis, risk treatments) need to be specified by other means.

- Finally, the absence of identified problems in the question-based or interaction analyses may produce a false sense of security. The users have to understand that this is only a helper to find application threats and not a substitute for their expertise and common sense.

8.7.3 Possible application

The proposed solution may be applied during the design risk assessment phase of an application software development lifecycle. There are no restrictions on the nature of applications under analysis, but the most profitable would be the complex enterprise applications handling various sensitive data (e.g. personally identifiable information). The primary auditory is security-educated staff responsible for application design: security auditors, analysts and software architects. For teams already using STRIDE or CORAS methods using Tam^2 should be the easiest.

8.8 Summary

In the context of this chapter we identified and tried to address some limitations of the current popular threat modeling approaches, in particular STRIDE. As already mentioned in the previous sections of this deliverable, the major drawbacks of such methods are (1) imprecise informal specification of architectural models; (2) laboriousness of threat and vulnerability identification; and (3) one-sided system view (either software-centric or asset-centric).

ESCUDO-CLOUD contributes to the Tam^2 threat modeling approach, based on STRIDE, TAM, and CORAS, that permits to (1) focus the analysis on protecting the most important assets; (2) partly automate the threat identification; and (3) document the high-level risk assessment results. We sought a fine balance between the user-friendly informal approaches and powerful yet heavy-weight solutions based on formal methods. We showed how enriching architecture diagrams with some security-related information can enable a lightweight (automated) interaction analysis and reduce the scope of a checklist-based analysis. Finally, we developed a proof-of-concept prototype that was used to illustrate the presented ideas on a general case study throughout this chapter.

9. Conclusions

In this document, we have developed an innovative, requirement-based threat analysis approach applicable for the ESCUDO-CLOUD Use Cases. Each Use Case addresses a specific application scenario classified as Infrastructure Provisioning, Cloud Storage, and Cloud Processing. This document explored the potential for conducting threat analysis at the requirements level. Specifically, it focused on threats that can potentially violate the user's data ownership requirements of security, functionality and performance.

The progression of the threat analysis activities was the following. We first performed the initial requirement-based threat analysis by analyzing the Use Case requirements as obtained from D1.1/1.2. These requirements were then analyzed from the security perspectives along with extracting the threat implication that can occur from the violation of these requirements. Subsequently, we analyzed the requirements of each Use Case, the assumptions behind each requirement and their potential impact on security, performance and functionality. As an ESCUDO-CLOUD innovation, we developed a novel hierarchical process to analyze the requirements by exploring both direct and indirect assumptions behind these requirements, and also their dependencies on other requirements. Such a dependency analysis is critical for evaluating the risks associated with inter-dependent requirements. After analyzing each Use Case, we cross examined all four Use Cases to ascertain the common requirement required by all Use Cases. The extracted common requirements were grouped together into logical categories along with outlining the estimated likelihood of violation in each category and the consequent impact on threat severity. Using this developed approach, we have shown the utility of RBTA to be applicable to all ESCUDO-CLOUD Use Cases.

Overall, the deliverable covered the (a) development of a novel dependency representation model for validating each Use Case requirements, (b) identification of the threat severity likelihoods across the Use Case levels and also illustrated the areas where security mechanisms should be applied to protect the services, and (c) developed a novel semi-automated security threat assessment approach based on the system security objectives and a high-level design enriched with security aspects. Thus, D2.4 has established the effectiveness of the RBTA approach for the ESCUDO-CLOUD Use Cases.

Bibliography

[AS/04] AS/NZS4360. Australian/New Zealand Standard for Risk Management, 2004. Standards Australia/Standards New Zealand.

[Bor11] M. Borozdin. Architecture-Based Threat Analysis. Master's thesis, University of Trento, 2011.

[Bro01] T. Browning. Applying the design structure matrix to system decomposition and integration problems: a review and new directions. *IEEE Transactions on Engineering Management*, 48(3):292–306, 2001.

[CAP] Common Attack Pattern Enumeration And Classification. `http://capec.mitre.org/`.

[CER] United States Computer Emergency Readiness Team. `http://www.us-cert.gov/`.

[Com] Common Criteria for Information Technology Security Evaluation. `http://www.commoncriteriaportal.org/cc/`.

[CVS] Common Vulnerability Scoring System. `http://www.first.org/cvss`.

[DWTB03] Y. Deng, J. Wang, J.P. Tsai, and K. Beznosov. An approach for modeling and analysis of security system architectures. *IEEE Transactions on Knowledge and Data Engineering*, 15(5):1099–1119, 2003.

[GE91] D. Gebala and S. Eppinger. Methods for analyzing design procedures. In *Proc. of Design Theory and Methodology*, pages 227–233, 1991.

[Hea07] E. I. Heidi and Dahl et al. Structured semantics for the CORAS security risk modelling language. In *Pre-proceedings of the 2nd International Workshop on Interoperability solutions on Trust, Security, Policies and QoS for Enhanced Enterprise Systems (IS-TSPQ'07)*, pages 79–92, 2007.

[HL06] M. Howard and S. Lipner. *The Security Development Lifecycle*. Microsoft Press, 2006.

[HLOS06] S. Hernan, S. Lambert, T. Ostwald, and A. Shostack. Uncover Security Design Flaws Using The STRIDE Approach. *MSDN Magazine*, 2006.

[HM04] G. Hoglund and G. McGraw. *Exploiting Software How to Break Code*. Addison Wesley, 2004.

[Ing16] T. R. Ingoldsby. Attack Tree-based Threat Risk Analysis. `http://www.amenaza.com/downloads/docs/AttackTreeThreatRiskAnalysis.pdf`, March 2016.

[ISO] ISO 31000 - Risk management. `http://www.iso.org/iso/home/standards/iso31000.htm`.

[Jac12] D. Jackson. Alloy: a language and tool for relational models. `http://alloy.mit.edu/alloy/`, 2012.

[Jür02] J. Jürjens. UMLsec: Extending UML for Secure Systems Development. In *Proceedings of the 5th International Conference on The Unified Modeling Language*, pages 412–425, 2002.

[JW98] G. Jelen and J. Williams. A practical approach to measuring assurance. In *Proc. of Computer Security Applications Conference*, pages 333–343, 1998.

[LBD02] T. Lodderstedt, D. A. Basin, and J. Doser. SecureUML: A UML-Based Modeling Language for Model-Driven Security. In *Proceedings of the 5th International Conference on The Unified Modeling Language*, pages 426–441, 2002.

[LH05] S. Lipner and M. Howard. The Trustworthy Computing Security Development Lifecycle. `http://msdn.microsoft.com/en-us/library/ms995349.aspx`, 2005.

[Lip04] S. Lipner. The Trustworthy Computing Security Development Lifecycle. In *Proceedings of the 20th Annual Computer Security Applications Conference*, pages 2–13, 2004.

[LSS11] M. S. Lund, B. Solhaug, and K. Stølen. *Model-Driven Risk Analysis - The CORAS Approach*. Springer, 2011.

[McG06] G. McGraw. *Software Security: Building Security In*. Addison-Wesley Professional, 2006.

[MCY99] J. Mylopoulos, L. Chung, and E. Yu. From object-oriented to goal-oriented requirements analysis. *Commun. ACM*, 42(1):31–37, 1999.

[Mei03] J.D. Meier. Improving Web Application Security: Threats and Countermeasures. `http://msdn.microsoft.com/en-us/library/ff648644.aspx`, 2003.

[MEL01] A. P. Moore, R. J. Ellison, and R. C. Linger. Attack Modeling for Information Security and Survivability, 2001. www.cert.org/archive/pdf/01tn001.pdf.

[Mica] Microsoft Exploitability Index. `http://technet.microsoft.com/en-us/security/cc998259`.

[Micb] Security Bulletin Severity Rating System. `http://technet.microsoft.com/en-us/security/gg309177.aspx`.

[MM87] D. Marca and C. McGowan. SADT: structured analysis and design technique. In *McGraw-Hill*, 1987.

[NVDa] National Vulnerability Database. `http://nvd.nist.gov/`.

[NVDb] Security Controls and Assessment Procedures for Federal Information Systems and Organizations. `http://web.nvd.nist.gov/view/800-53/home`.

[OSC06] E. A. Oladimeji, S. Supakkul, and L. Chung. Security Threat Modeling and Analysis: A goal oriented approach. In *Software Engineering Applications*, 2006.

[Ros77] D. Ross. Structured analysis (SA): A language for communicating ideas. In *Software Engineering*, number 1, pages 16–34, 1977.

[Sch99] B. Schneier. Attack Trees - Modeling security threats. `http://www.schneier.com/paper-attacktrees-ddj-ft.html`, 1999.

[SDL] SDL Threat Modeling Tool. `http://www.microsoft.com/security/sdl/adopt/threatmodeling.aspx`.

[SHCO10] S. Supakkul, T. Hill, L. Chung, and E. A. Oladimeji. Goal-oriented security threat mitigation patterns: a case of credit card theft mitigation. In *Proceedings of the 16th Conference on Pattern Languages of Programs*, pages 17:1–17:15, 2010.

[Sho] A. Shostack. Experiences Threat Modeling at Microsoft. `https://adam.shostack.org/modsec08/Shostack-ModSec08-Experiences-Threat-Modeling-At-Microsoft.pdf`.

[SJSJ05] N. Sangal, E. Jordan, V. Sinha, and D. Jackson. Using dependency models to manage complex software architecture. *Sigplan Notices*, 40(10):167–176, 2005.

[SO05] G. Sindre and A. L. Opdahl. Eliciting security requirements with misuse cases. *Requir. Eng.*, 10(1):34–44, 2005.

[Ste81] D. Steward. The design structure system: a method for managing the design of complex systems. *IEEE Transactions on Engineering Management*, (3):71–74, 1981.

[SWSS09] Christian Sell, Matthias Winkler, Thomas Springer, and Alexander Schill. Two dependency modeling approaches for business process adaptation. In *International Conference on Knowledge Science, Engineering and Management*, pages 418–429. Springer, 2009.

[TAMa] Microsoft Threat Analysis and Modeling Tool. `http://www.microsoft.com/en-us/download/details.aspx?id=14719`.

[TAMb] TAM - The SAP way combining FMC and UML. `http://www.fmc-modeling.org/fmc-and-tam`.

[Thr] Michael Howard Teaches Threat Modeling. `http://www.microsoft.com/en-us/download/details.aspx?id=24617`.

[TMT$^+$16] A. Taha, P. Metzler, R. Trapero, J. Luna, and N. Suri. Identifying and utilizing dependencies across cloud security services. In *Proc. of Asia Conference on Computer and Communications Security*, pages 329–340, 2016.

[VM01] J. Viega and G. McGraw. *Building Secure Software: How to Avoid Security Problems the Right Way*. Addison-Wesley Professional, 2001.

[Wei05] S. Chun Wei. Misuse Cases and Abuse Cases in Eliciting Security Requirements. `http://www.cs.auckland.ac.nz/courses/compsci725s2c/archive/termpapers/csia.pdf`, 2005.

[WL77] J. Wiest and F. Levy. A management guide to PERT/CPM. In *Prentice-Hall*, 1977.

[WS09] M. Winkler and A. Schill. Towards dependency management in service compositions. In *Proc. of e-Business*, pages 79–84, 2009.

[WSS10] M. Winkler, T. Springer, and A. Schill. Automating composite sla management tasks by exploiting service dependency information. In *Proc. of Web Services*, pages 59–66, 2010.

[XN06] D. Xu and K. E. Nygard. Threat-Driven Modeling and Verification of Secure Software Using Aspect-Oriented Petri Nets. *IEEE Transactions on Software Engineering*, 32:265–278, 2006.

[ZJRR12] Y. Zhang, A. Juels, M. Reiter, and T. Ristenpart. Cross-VM side channels and their use to extract private keys. In *Proc. of Computer Communication and Security*, pages 305–316, 2012.