BACHELOR THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

CoReTM 2.0: a Design Study on Semi-automated Threat Modeling with STRIDE-per-Interaction

Miro Valentino Vannini
Zürich, Switzerland
Student ID: 18-926-063

Supervisor: Jan von der Assen, Jürgen Messerer, Thomas Grübl
Date of Submission: September 17, 2024

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

Bachelor Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Abstract

Cyberangriffe stellen eine erhebliche Bedrohung für Unternehmen und ihre technische Infrastruktur dar. Um diesen Bedrohungen entgegenzuwirken, setzen Unternehmen verschiedene Gegenmassnahmen ein, darunter auch die Bedrohungsmodellierung. Dieser proaktive Ansatz wird während der Planungsphase eines Systems angewandt, um potenzielle Bedrohungen frühzeitig im Entwicklungsprozess zu identifizieren. Auf diese Weise können Unternehmen Strategien entwickeln, um diese Schwachstellen in ihren Systemen zu entschärfen und letztlich Kosten zu sparen, die mit der Behebung dieser Probleme in einem späteren Stadium verbunden sind. In dieser Arbeit wird die Entwicklung und Implementierung von *CoReTM 2.0* vorgestellt, einem halbautomatischen Tool zur Bedrohungsmodellierung unter Verwendung der STRIDE-per-Interaction Methode. Um die Herausforderung der manuellen Identifizierung von Bedrohungen in Systeminteraktionen zu bewältigen, bettet *CoReTM 2.0* Draw.io ein, sodass Benutzer*innen Datenflussdiagramme (DFDs) erstellen können. Das Tool analysiert diese Diagramme, um strukturierte Bedrohungsmodelle zu generieren. Diese Generierung umfasst die automatische Erstellung einer Übersichtstabelle auf der Grundlage von Bedrohungen ausgesetzten Systeminteraktionen. Nachdem die Übersichtstabelle manuell ausgefüllt wurde, wird sie vom System analysiert und die Bedrohungstabellen werden generiert. *CoReTM 2.0* reduziert den manuellen Aufwand für die STRIDE-per-Interaction-Analyse erheblich. Diese Arbeit beschreibt die Architektur, das Design und die Implementierung von *CoReTM 2.0* und hebt seine Fähigkeit hervor, den Prozess der Bedrohungsmodellierung zu unterstützen. Die mit einem Industriepartner durchgeführte Evaluierung ergab keine Fehler und eine hohe Benutzerzufriedenheit, was sich in einer hohen Punktzahl auf der System Usability Scale (SUS) widerspiegelt. Damit wurde gezeigt, dass die Bedrohungsmodellierung mit der STRIDE-per-Interaction Methode halbautomatisch erfolgen kann und Nutzer*innen effektiv durch den Prozess geführt werden. Trotz des Erfolgs wurden zusätzliche Funktionen, wie erweiterte Exportoptionen und eine verbesserte Darstellung von Texteingabefeldern, als notwendig für die Anwendung in der Praxis identifiziert. Der Open-Source-Charakter von *CoReTM 2.0*, lizenziert unter Apache-2.0, fördert künftige Beiträge und Anpassungen und legt damit den Grundstein für weitere Fortschritte in der halbautomatisierten Bedrohungsmodellierung unter Verwendung der STRIDE-per-Interaction Methode.

Cyberattacks pose a significant threat to companies and their technical infrastructure. To mitigate these threats, companies employ various countermeasures, one of which is threat modeling. This proactive approach is implemented during the system design phase to identify potential threats early in the development process. By doing so, organizations can develop strategies to mitigate these vulnerabilities in their systems, ultimately saving costs associated with fixing these issues at a later stage. This thesis presents the development and implementation of *CoReTM 2.0*, a semi-automated tool for threat modeling using the STRIDE-per-Interaction methodology. Addressing the challenge of manual threat identification in system interactions, *CoReTM 2.0* embeds Draw.io to enable users to create Data Flow Diagrams (DFDs). The tool then parses these diagrams to generate structured threat models. This generation involves automatically creating an overview table based on system interactions exposed to threats. After the user manually fills out the overview table, the system analyzes it and generates the threat tables. *CoReTM 2.0* significantly reduces the manual effort required for the STRIDE-per-Interaction analysis. This thesis outlines the architecture, design, and implementation of *CoReTM 2.0*, focusing on its ability to simplify the threat modeling process. The evaluation, conducted with an industrial partner, revealed no bugs and high user satisfaction, as reflected in a high System Usability Scale (SUS) score. It has been demonstrated that threat modeling using the STRIDE-per-Interaction methodology can be semi-automated, guiding the user through the process effectively. Despite its success, further features such as enhanced report export capabilities, improved text field display, and more are identified as necessary for real-world application. The open-source nature of *CoReTM 2.0* licensed under Apache-2.0 encourages future contributions and adaptations, laying the groundwork for further advancements in semi-automated threat modeling using the STRIDE-per-Interaction methodology.

Acknowledgments

I want to express my gratitude to my supervisors for giving me the opportunity to write this thesis and for their support throughout the process. I especially want to thank Jan von der Assen for his guidance and for providing valuable feedback at every stage of the thesis creation, as well as establishing the initial theoretical framework of CoReTM. This thesis wouldn't have been possible without his input. I also want to extend my thanks to our industrial partner bbv, and particularly to Jürgen Messerer, for their willingness to evaluate the practical aspects of this work and for providing requirements that greatly contributed to the definition of the architecture and implementation planning phase.

Contents

Abstract **i**

Acknowledgments **iii**

1 Introduction **1**

1.1 Motivation 1

1.2 Description of Work 1

1.3 Thesis Outline 2

2 Background **3**

2.1 Threats and Vulnerabilities 3

2.2 Security Engineering 3

2.3 Threat Modeling 4

2.3.1 STRIDE 5

2.3.2 STRIDE-per-Interaction 7

3 Related Work **9**

3.1 Tools 9

3.2 Comparative Analysis Conclusion 12

4 Architecture **13**

4.1 Requirement Analysis for Minimum Viable Product . . . 13

4.1.1 Modeling . . . 14

4.1.2 System . . . 14

4.1.3 Threat Analysis . . . 15

4.2 Architectural Diagram . . . 16

4.3 Further Features and Changes . . . 17

5 Design & Implementation **19**

5.1 Technology Stack . . . 19

5.2 Components . . . 20

5.2.1 Views . . . 21

5.2.2 Interfaces and Storage . . . 22

5.2.3 Modeling and Parsing . . . 26

5.2.4 Tables . . . 31

5.3 Demonstration . . . 34

5.4 Second Development Iteration . . . 37

5.4.1 Minor Changes . . . 37

5.4.2 Major Changes . . . 39

6 Evaluation **43**

6.1 Questionnaire . . . 43

6.2 Results . . . 45

6.3 Findings . . . 49

7 Summary **51**

7.1 Future Work . . . 52

Bibliography **53**

Abbreviations **59**

List of Figures **59**

List of Tables **61**

List of Listings **63**

A System Usability Scale **67**

B Installation Guidelines **71**

C Contents of the CD **73**

Chapter 1

Introduction

In today's digital age, cyber threats are increasingly relevant. The number of cyberattacks and the damage they cause is rising due to new tools and techniques [1]. According to the Global Cybersecurity Outlook 2024 report, cyberattacks are not only affecting big companies but also small businesses. The report states that 29% of the organizations surveyed experienced some form of cyber incident in the past 12 months, posing a significant risk to these companies [2]. To address these threats, one approach is through threat modeling, which is crucial for identifying and mitigating security risks during the design phase of system development [3]. There are various methodologies for threat modeling, with one popular one being STRIDE. This methodology allows for security analysis even for individuals with limited cybersecurity knowledge [4]. Despite its simplicity, threat modeling using the STRIDE methodology involves significant manual work. This thesis introduces *CoReTM 2.0*, a new tool designed to semi-automate the STRIDE-per-Interaction methodology, making threat modeling more accessible and efficient.

1.1 Motivation

The motivation behind this work is to develop a tool that enables the semi-automated application of the STRIDE-per-Interaction methodology. By creating *CoReTM 2.0*, this thesis aims to simplify the threat modeling process and make it accessible to a broader audience in an open-source format. A key aspect of this solution is its full transparency, as it openly discloses all underlying functionalities, allowing users to verify that the threat modeling processes uphold the rigor necessary for thorough system security evaluations. This tool is also designed to serve as a foundation or inspiration for future development and research in the field, encouraging to build upon or extend its capabilities.

1.2 Description of Work

The thesis argues that no tools are currently available that semi-automate the implementation of the STRIDE-per-Interaction methodology while also being open-source and free

to use. Once this is validated, the thesis will develop software based on the requirements from an industrial partner to demonstrate the feasibility of such semi-automation. This implementation will be based on the prototype of CoReTM [5], [6]. By achieving this, this work contributes to the threat modeling domain in the form of a new tool that everyone can use. Furthermore, based on an expert-based evaluation, it showcases that it is possible to semi-automate some work within threat modeling that used to be done manually, and thereby might inspire other developers to use it or its underlying logic to craft their own tools with similar functionality for different methodologies.

1.3 Thesis Outline

This work follows a design study structured to guide the reader through a logical progression of concepts and findings. In Chapter 2, the reader is introduced to critical background information essential for understanding the topic and becoming familiar with the relevant terminology. The existing research in the field is examined in Chapter 3, providing an overview of current developments and identifying the research gap that this work addresses. Following this, the author presents the architecture of the software developed as the core of this study in Chapter 4, offering a high-level explanation of its components and the overall system flow. Chapter 5 delves into the critical implementation details of the application. Chapter 6 assesses the application using various evaluation strategies to determine its performance and effectiveness. The work concludes with a summary in Chapter 7, reflecting on the contributions and outcomes.

Chapter 2

Background

This chapter explains the foundational concepts and methodologies necessary to understand this work's subsequent sections by covering the critical aspects of information security and the nature of threats and vulnerabilities that challenge IT systems' integrity, confidentiality, and availability. This chapter emphasizes the importance of secure software development by exploring security engineering principles, including established concepts like Microsoft's Security Development Lifecycle (SDL) [7]. Furthermore, the chapter delves into the concept of threat modeling, presenting the STRIDE methodology by using an example.

2.1 Threats and Vulnerabilities

A threat refers to any action or circumstance that could lead to an undesired impact on an IT system, including its assets and resources. The impact can take different forms, such as alteration, destruction, unauthorized disclosure, infiltration, etc. Any scenario that may affect the CIA triad, which includes *Confidentiality, Integrity, and Availability*, is considered a threat [8], [9].

A vulnerability is a defect or weakness in a system – due to design flaws, configuration errors, or insecure coding practices – that can be exploited by a threat. Vulnerabilities can occur at any system level, including software, hardware, network, or human actions. This could include issues like weak input validation, which can lead to attacks. To summarize, a threat is a potential event that can adversely affect an asset, whereas a successful attack exploits vulnerabilities in a system [10].

2.2 Security Engineering

According to Microsoft's SDL, creating secure software is of utmost importance, not only from a customer's perspective, who is put at risk of attacks with every security vulnerability in a system but also for the company/vendor of the software. Creating secure

software is not free; money, time, and effort are needed to develop software following the SDL guidelines [7], [11]. SDL consists of five different phases, as seen in Figure 2.1. This work predominantly focuses on the threat modeling that takes place in the *Design* phase of the SDL, specifically delving into the STRIDE methodology. It thoroughly examines STRIDE's strategies and implications, prioritizing its comprehensive analysis over other frameworks and methodologies. In contrast, the other four phases of SDL will receive less extensive coverage.

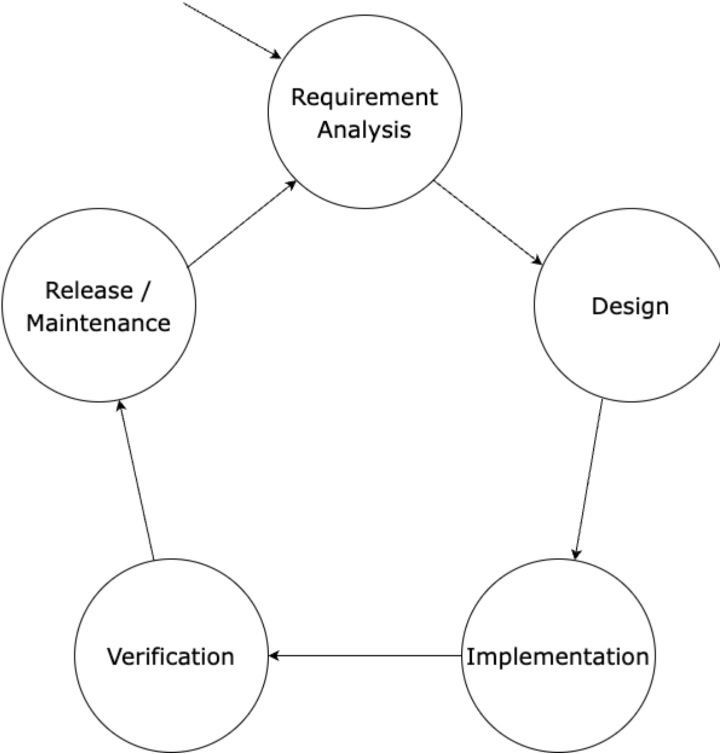


Figure 2.1: Security Development Lifecycle Process Diagram

2.3 Threat Modeling

Threat modeling is the process of identifying threats and vulnerabilities in a system and taking measures to prevent threats from occurring. This process is usually carried out during the design phase but can also be done when the system is already operational [4]. It is highly advantageous to find vulnerabilities early on as it allows for mitigating threats before they can even occur. Conducting a threat modelization is very cost-intensive since it requires information security and other departments' cooperation [12]. Therefore, it makes sense to do it upfront to ensure the engineers can deliver a better product and to omit multiple iterations.

Usually, the threat modeling process is divided into three phases [13]: *(i)* identifying assets and access points, *(ii)* listing all potential threats, and *(iii)* building a mitigation plan.

(i) **Identifying assets and access points:** Assets refer to valuable possessions owned by individuals or companies. These can be targeted by malicious actors seeking to gain access, control, or destroy them. Identifying these assets is a crucial first step in threat modeling, as they are the primary objectives of potential threats. Access points, on the other hand, are parts of a system through which attackers can try to interact with and gain entry into the system to reach these assets. Examples of access points include login interfaces, file systems, and hardware points. By identifying these access points, trust boundaries are established within the system. These boundaries determine the different levels of trust required to access various components of the system [4].

(ii) **Listing all potential threats:** Now that the system has been analyzed, it's time to use the insights gained and identify potential threats. This involves reviewing all the assets, entry points, and their interactions. To do this, this work will focus on the STRIDE model, which will be explained in detail in the next section.

(iii) **Building a mitigation plan:** Finally, once the threats to which the assets are exposed have been identified and understood, a mitigation plan is proposed.

2.3.1 STRIDE

STRIDE is a threat-centric modeling methodology developed by Microsoft in the late 90s that identifies threats using a mnemonic for six different types of threats [14]. It is the most widely accepted threat modeling process and allows people with little knowledge in cybersecurity to apply it [4], [13]. Unlike other approaches, STRIDE's focus shifts from identifying all possible attacks to the results of such possible attacks. This point of view allows the user to reason what an attacker aims to achieve by exploiting a vulnerability rather than thinking about numerous specific attacking strategies and techniques [15]. The six different types of the mnemonic are as follows [4]:

(i) ***S****poofing:* Includes activities in which a person or a program masquerades itself and pretends to be someone or something else. A common example is email spoofing, where the attacker sends an email with a forged sender address.

(ii) ***T****ampering:* Tampering refers to changing a system, network, or data, including adding, modifying, or deleting entire components. An example of tampering could be modifying a file on a server.

(iii) ***R****epudiation:* When someone denies having done something or being responsible for something that happened, this threat falls into the category of repudiation. In some cases, individuals may try to attack the system logs to prevent verification of their involvement or the malicious activity.

(iv) ***I****nformation Disclosure:* Is understood as disclosing information to someone who is not allowed to see it. This can include gaining insights through *e.g.*, error messages, file names, reading data from a network, etc.

(v) ***D**enial of Service (DoS):* DoS attacks are designed to prevent a resource from functioning normally. This can be achieved by *e.g.*, flooding the network with requests or filling up a process's memory. These attacks aim to occupy the resource to the point where it cannot operate effectively.

(vi) ***E**levation of Privilege:* Similar to information disclosure, threats of elevation of privilege allow someone to do something they are not authorized to. This can happen when *e.g.*, authorization checks can be bypassed, and the attacker can execute code as admin or similar.

Since the STRIDE model is used to identify threats, the user can derive possible structural vulnerabilities and develop a mitigation plan. According to [13], the system to be modeled should be assessed by identifying assets and access points in the first step. This is done using so-called DFDs. DFDs are a graphical way of describing a system, showing all the inputs and outputs, and representing all the internal logical processes and components [16]. It consists of six different symbols: *Data Flow* (One-way arrow), *Data Store* (Two parallel horizontal lines), *Process* (Circle), *Multiprocess* (Two concentric circles), *Interactors* (Rectangle) and *Trust Boundary* (Dotted line) [17]. The described symbols can be observed in Figure 2.2. Secondly, the threats derived by the DFD are discussed. These threats are then recorded in a table. Once all the threats have been identified, a mitigation plan is developed to address each of them. The goal of the plan is to build the system in a way that eliminates or minimizes the identified threats. In the next section, an example of STRIDE is provided.

Symbol	Name	Explanation
	Data Store	Things that store data
	Process	Any running code
	Multiprocess	Many processes
	Data Flow	Communication between elements
	Interactor	External people or code outside your control
	Trust Boundary	Border between trusted and untrusted elements

Figure 2.2: DFD Symbols extension to [4]

2.3.2 STRIDE-per-Interaction

A variant of STRIDE, also developed by Microsoft, is STRIDE-per-Interaction. This approach focuses on how threats show up in the interaction of system components rather than in isolation. Therefore, the focus lies on how the components interact and what kind of threats could exist in these interactions. To point the threats out, a tuple is used in the form of *(origin, destination, interaction)* and the threats are enumerated. STRIDE-per-Interaction is often realized using an overview table and threat tables [4].

A simplified example from [4] is analyzed to understand how STRIDE-per-Interaction operates. Assuming a system comprising a browser, a process, and a database as shown in Figure 2.3.

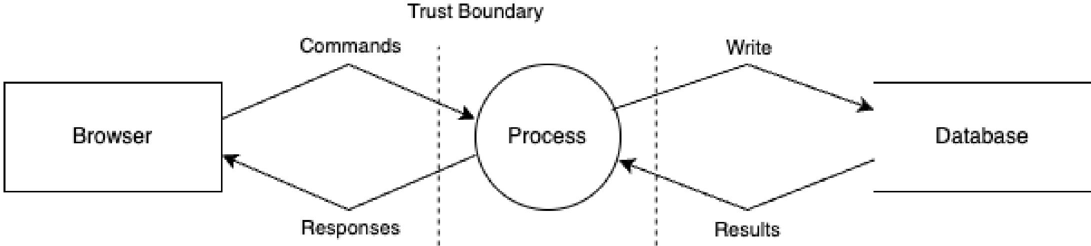


Figure 2.3: DFD of an Example System

An overview table is used to clarify the threats associated with each interaction. Table 2.1 includes columns for # (number for line reference), Element, Interaction, and applicable STRIDE threats to the interaction. In this example, we can see that there is a threat of spoofing in the process component, indicated by the 'x' in the 'S' column. Note that Table 2.1 provides an overview of the threats without going into detail. However, it should be sufficient to understand the STRIDE-per-Interaction process. Once the user has created the detailed threat tables with elaborated threats put into words, they can begin to define a mitigation plan for the identified threats.

Table 2.1: STRIDE-per-Interaction: Threat Applicability Overview

#	ELEMENT	INTERACTION	S	T	R	I	D	E
1	Process	Process has outbound data flow to data store.	x			x		
2		Process has inbound data flow from data store.	x	x			x	x
3		Process sends output to external interactor (code).	x		x	x	x	
4		Process sends output to external interactor (human).			x			
5		Process has inbound data flow from external interactor.	x				x	x
6	Data Flow	Crosses machine boundary.		x			x	x
7	Data Store	Process has outbound data flow to data store.		x	x	x	x	
8		Process has inbound data flow from data store.	x			x		
9	External Interactor	External interactor passes input to process.	x		x	x		
10		External interactor passes input to process.	x					

Chapter 3

Related Work

This section discusses and elaborates on different commonly used tools to determine whether they are suitable for using STRIDE-per-Interaction, as presented in the previous chapter. The goal is to find a tool that offers the user the most flexibility, meaning that it fulfills most of the following dimensions:

Q1) Does it allow the creation of a DFD?

Q2) What threat modeling methodologies are applicable within the tool?

Q3) What is the underlying revenue model?

Q4) Where does the software operate?

Q5) Is the tool cross-platform?

Q6) To what degree of openness is the software published?

Generally, the tool that allows for most of these dimensions should win the race. However, some dimensions are more important than others. Therefore, they are ordered according to their importance.

3.1 Tools

The following section will introduce various tools closely associated with threat modeling. These tools will be discussed in detail, highlighting their key features and benefits. Since many different tools exist, covering all of them would be out of scope for this work; therefore, only tools that claim to integrate the STRIDE methodology and allow the creation of DFDs in a way are reviewed. Furthermore, this analysis only contains well-established threat modeling tools or ones that originate from an academic background. To conclude the comparative analysis, Table 3.1 provides the findings of the dimensions, presenting a

Table 3.1: Comparison of Different Threat Modeling Tools

	DFD	Methodology	Revenue Model	Operation	Cross Platform	Openness
CAIRIS	✓	IRIS	Free	On-Premise	✓	open
CoReTM	✓	SpI*, PASTA	Free	?	✓	open
IriusRisk	✓	SpE**, OCTAVE, TRIKE, PASTA	Licensing	On-Premise & Managed	✓	?
MTMT	✓	SpE**	Free	On-Premise	✗	closed
OWASP TD	✓	SpE**, LINDDUN, CIA, DIE, PLOT4ai	Free	On-Premise	✓	open
Threagile	partially	rule based	Free	On-Premise	✓	open

* STRIDE-per-Interaction, ** STRIDE-per-Element

comprehensive and clear overview of the results. Finally, some models allow the application of methodologies other than STRIDE. These methodologies will be mentioned but not elaborated on further.

CAIRIS stands for Computer Aided Integration of Requirements and Information Security. It is an open-source platform with many different features. There is a client version and a deprecated desktop application. Users can manually create DFDs [18] or import existing ones using their REST API or accepted formats as from *e.g.*, Draw.io [19], [20]. Furthermore, CAIRIS supports other modeling approaches, such as top-down representations to attack trees [3]. Although explicitly designed for the IRIS framework, CAIRIS aimed to become methodology agnostic [21]. However, it does not have the functionality to automatically create the necessary tables for the Stride-per-Interaction methodology out of the box. But since the software is open-source and free to use, it can be extended by building this feature one-self [22]. CAIRIS is a self-hosted application and can be run using Docker or a REST Server [23].

A further tool that relies on DFDs is **CoReTM**, which is a novel approach to enable automated threat modeling in an annotation-based collaborative setting. Furthermore, it allows on-site, remote, or hybrid modeling with synchronous and asynchronous contributions. By wrapping Draw.io into a web application, users can generate DFDs and automatically create the STRIDE-per-Interaction table. Two prototypical but nonfunctional implementations exist on GitHub [6], [24]. Therefore, further information is derived from the proposal paper. *CoReTM* allows the application of various threat modeling methodologies, such as STRIDE-per-Interaction or PASTA, and is open and accessible to anyone. When creating a new model, the user is guided according to the chosen methodology, which is decided upfront and navigated through a questionnaire afterward. This ensures usability, even for non-technical users [5], [25]. There is currently no information available about the deployment and maintenance of the software since there is no productive application. However, if implemented as described, it would be an open-source web application meant to be cross-platform and without any operating system dependencies.

In contrast, **IriusRisk** is a well-established threat modeling tool that has been on the market for multiple years. It is a pattern-based threat modeling tool that allows the mapping of discovered threats to a mitigation plan automatically. Although IriusRisk can suggest countermeasures based on identified threats, users can input and manage their countermeasures using the platform's pattern editor [26]. IriusRisk has three different pricing plans, *Community, Enterprise Pro,* and *Professional*, with varying capabilities. In the open-source community tier [27], users can create one threat model and use analytics features but cannot collaborate in a team [28]; the other licensing tiers are closed-source.

Therefore the cell in Table 3.1 is marked with a '?'. Furthermore, IriusRisk offers the usage of multiple threat-modeling methodologies out of the box [29], including STRIDE-per-Element, which is applied using a questionnaire [30]. The embedding of Draw.io allows the creation of custom DFDs. IriusRisk has a threat library covering common knowledge bases and allowing the user to create self-defined threats [3]. IriusRisk can be obtained as a managed service or on-premise and is not platform dependent [31].

Another well-established tool in the market is the **Microsoft Threat Modeling Tool** (MTMT). It is a free software provided by Microsoft. Several references found in different industries applied this tool to do threat modeling (*cf.*, [13], [32]–[35]). It allows the creation of detailed DFDs with extensive configuration options and attribute settings for the elements [36]. It follows an integrated STRIDE-per-Element [37] methodology. The STRIDE-per-Element approach differs from the one presented by considering each system component in isolation rather than analyzing the end-to-end scenarios where several components interact. The STRIDE-per-Interaction approach is considered to have a wider scope by considering the threats that might occur in the interaction of different software components and is, therefore, more informative [38]. Even though STRIDE-per-Element is supported, the guided analysis provided by MTMT does not fit the STRIDE-per-Interaction methodology. Microsoft is actively managing and maintaining the software (last release in October 2023 [39]). Even though earlier versions of MTMT used to be open-source, the source code of the current application version TMT7 could not be found. Therefore, it is considered closed-source. Furthermore, a noteworthy mention is that the software is only available for Microsoft OS and excludes other operating systems.

In comparison, **OWASP Threat Dragon** (TD) is an open-source threat modeling application that supports multiple methodologies. It is available as a web and desktop application for Linux, Mac OS, and Windows and supports 10 different languages. The application allows the creation of DFDs and has a built-in rule engine to auto-generate threats and mitigation strategies [40]. Although STRIDE is a supported methodology, the creation of the threats is not summarized in the form of the desired tuple, as required in STRIDE-per-Interaction. Thus, STRIDE-per-Interaction is not integrated fully into OWASP TD. Due to the simplicity of OWASP TD, the application is very accommodating to less experienced developers and, therefore, more user-friendly as *e.g.*, MTMT with its extensive configuration possibilities [36], [41].

Finally, **Threagile** is an open-source toolkit for agile threat modeling. It was first published in 2020 and is still actively maintained [42]. The tool lets users model a system's architecture directly through a YAML file. When run, Threagile applies a set of risk rules to the architecture and generates a report outlining potential risks and mitigation strategies for constructing a DFD. Due to the automatic generation of a risk report, Threagiles methodology is classified as *rule based* in Table 3.1. It's important to note that Threagile doesn't support manual DFD modeling; instead, users need to specify it in the YAML file [43]. Therefore, it is noted as partially given in Table 3.1. Additionally, since the report is automatically generated based on the risk rules, direct application of the STRIDE methodology is not supported. However, as an open-source project [42], users can create their own STRIDE extension, although this would require some familiarity with the application's stack and programming skills. Threagile can be executed in various ways, including via the command line (or Docker) or as a REST-Server [43].

3.2 Comparative Analysis Conclusion

Based on the conducted analysis of existing tools, the initial questions can be summarized as follows. Regarding (*Q1*), it can be observed that DFDs are widely adopted by various threat modeling tools. Even though only tools with this capability were considered, the fact that several could be found integrating DFDs as one core functionality is evidence of the popularity of DFDs. While this does not directly present a new avenue for future research, it confirms that DFDs are a practical, generic means to capture the architectural abstractions of systems. Thus, new solutions and methods could rely on this feature since users are likely familiar with it. Concerning the second question (*Q2*), the tools adopt a diverse pool of methodologies. With the pre-condition in the selection of the tools to review that STRIDE must be applicable in some way, it has been shown that no tool exists that supports an application of the STRIDE-per-Interaction method in a guided and semi-automated way. What poses a promising niche for future threat modeling tools. In many of the tools that have been reviewed, there is a common theme of embracing an open source (*Q6*), free (*Q3*), and cross-platform (*Q5*) approach. This can be attributed to several reasons. For example, it may be because the tool is extendable, has the potential to reach a larger user base by being platform-independent, allows contributions from people on the internet, or aims to be as transparent as possible to avoid mistrust. Finally, the trend regarding the deployment and maintenance of the tools (*Q4*) shows that only the paid option offers the possibility to run it as a managed service. In contrast, all the free tools require the user to host the software themselves. This makes sense from a perspective of trust and also financially since the free ones do not receive any monetary payment, and hosting servers would possibly not be within budget.

The research findings on various tools show that *CoReTM* has met most of the criteria set by other tools, making it the top choice. Although many of the reviewed tools are open-source and could, therefore, be extended to fulfill the requirements of supporting a semi-automated STRIDE-per-Interaction methodology by automatically creating the needed tables after modeling the system in the form of a DFD, this extension of the software comes with a certain cost. Furthermore, it would be a better contribution to the current situation if a working implementation of *CoReTM* is provided, creating a novel tool with inherent support for STRIDE-per-Interaction instead of STRIDE-per-Element, which is already present in many tools. The solution presented in this work will differ from the proposal of *CoReTM* in a few ways. In the prototype, there will be no synchronicity, which means that users cannot work on the same model and receive real-time updates. Additionally, the proposed prototype will only focus on the STRIDE-per-Interaction methodology and not on other methodologies like PASTA, which is mentioned in the *CoReTM* proposal [5]. The next parts of this work will address this matter by guiding the reader through architectural planning and outlining the essential specifications and requirements for *CoReTM 2.0*.

Chapter 4

Architecture

The upcoming chapter will cover the *CoReTM 2.0* framework and offer a high-level overview of the prototype that will be implemented later. It will start with a requirement analysis for a Minimum Viable Product (MVP) and highlight differences from the initial proposal. Then, the findings of the requirement analysis will be summarized in an architectural diagram, illustrating the major components and interfaces of the prototype. Specific implementation details and design aspects will be discussed in Chapter 5. This chapter aims to give the reader a basic understanding of "what the prototype consists of and what it is capable of doing" rather than "how the prototype is being built and designed".

The requirements that will be discussed were gathered from a design study [44] conducted with an industrial partner. They provided a list of requirements discovered during their research. Describing a comprehensive tool, including features that may not be feasible for this project's scope. As a result, the first step is to identify the fundamental requirements necessary for the tool to be usable. The evaluation takes place as soon as the MVP is implemented, followed by a second iteration of requirement analysis for further features.

4.1 Requirement Analysis for Minimum Viable Product

An MVP refers to a prototype that contains just enough features to be used by early users. Therefore, the focus of the features is clearly shifted onto the main utility the software must provide. These fundamental features are now discussed as requirements in the context of *CoReTM*, or rather *CoReTM 2.0*.

The process involves breaking down the requirements into smaller, independent units within the prototype's architecture instead of collecting them all simultaneously. This allows for a better understanding of each component's specific capabilities and responsibilities. Following the presentation of all essential requirements for the MVP, the outcomes are consolidated and documented in the subsequent sections and finally gathered in Table 4.1, which introduces the requirements according to three categories: functional modeling requirements, system aspects, and threat analysis.

Without diving into the specific technical details, the prototype implementation uses an embedding of Draw.io (also known as diagrams.net) for DFD-related tasks. The actual implementation of this will be discussed later. On a high level, Draw.io is a powerful and versatile open-source [45] software for creating various types of diagrams [46]. It offers an intuitive drag-and-drop interface, allowing users to place and connect shapes, text, and images easily. The tool supports real-time collaboration, integrates with cloud storage services like Google Drive, OneDrive, and Dropbox, and provides a variety of templates and shapes to streamline the diagram creation process. Diagrams can be exported and imported in various formats for easy sharing and embedding in documents or presentations [47].

4.1.1 Modeling

The following requirements focus on the essential modeling functionality needed to create customized DFDs within the *CoReTM 2.0* framework. It is crucial to be able to visually represent the system for threat modeling using the STRIDE-per-Interaction methodology. Therefore, this capability must be a core feature of the MVP, as it is paramount and represents the initial step in applying the STRIDE-per-Interaction method.

The prototype must provide several functionalities regarding DFDs. *1)* A set of custom stencils must be provided, consisting of the six different symbols present in DFDs. This is crucial to ensure the DFD is in the correct form. *2)* Users must be able to create elements within a modeling interface, and *3)* delete and move these elements around as needed. *4)* Each element in the DFD must have a unique ID or key to ensure proper identification and differentiation, which is important for the later diagram processing and table generation. *5)* It must be possible to connect already created elements through data flows. *6)* Each stencil element must be labelable, and *7)* these labels should be editable to accommodate any necessary modifications.

4.1.2 System

The system requirements focus on ensuring that the DFD creation and manipulation within the *CoReTM 2.0* framework is robust, user-friendly, and capable of supporting comprehensive threat modeling. The integration with Draw.io and the capability to manage trust boundaries are critical to achieving these goals. These requirements bridge the modeling and threat analysis requirements, ensuring seamless integration and functionality across the entire threat modeling process within the *CoReTM 2.0* framework. The focus is on creating a cohesive workflow from visual DFD creation to detailed threat analysis and documentation.

To accomplish this role, *1)* the system must integrate Draw.io. *2)* The system must also detect when a data flow crosses a trust boundary. *3)* It must recognize which elements are connected by a data flow and understand the orientation of this connection. *4)* To ensure accessibility, the system should be a web-based application, enabling anyone with internet access to use it. *5)* The system should allow users to export the created DFD and

6) tables in a meaningful format. This should happen in a fashion that does not rely on third-party applications or storage solutions, such that no sensitive data is compromised.

4.1.3 Threat Analysis

Finally, STRIDE-per-Interaction produces several tabular representations, which are then used to manually describe the threats and vulnerabilities of the system. This task is addressed within the threat analysis dimension, enabling the automation step in the *CoReTM 2.0* platform by automatically constructing the necessary overview and threat tables, saving users the effort of creating them manually. These tables support customization and note-taking, with text input fields and the option to add additional rows. They follow a strict design, making them suitable for automated creation and subsequent manual completion. The system requirements, which are responsible for parsing the DFD, enable the creation of automated tabular representations.

Several requirements must be met for these tables to fulfill their purpose. *1)* The generated overview table must have eight columns: Dataflow ID, Interaction (describing the elements in the DFD connected by the data flow), and columns for each STRIDE threat (S, T, R, I, D, E). This format ensures all necessary information is included, as detailed in Table 2.1. Checkboxes are present within the STRIDE columns. *2)* The user can tick and untick these checkboxes, defining if the corresponding STRIDE threat exists. *3)* Within the interaction column, the elements that are connected by a data flow must be displayed. *4)* Users must be able to describe the interaction in words by typing text into a text field. *5)* A threat table is created for each row in the overview table with at least one ticked cell in the STRIDE section. These threat tables contain columns for Threat ID, Dataflow ID, STRIDE Type, Threat, Mitigation, and Testing. *6)* The first three columns are automatically defined by the information provided in the overview table, *7)* while the remaining ones require user input. *8)* These cells are editable, and additional rows can be added to accommodate multiple threats, such as more than one possible tampering threat in a specific interaction.

Table 4.1: Overview Requirements for the MVP Prototype

Category	#	Requirement
Modeling	1	Custom stencils must be provided, consisting of DFD symbols.
	2	Creation of DFD elements.
	3	Deletion of DFD elements.
	4	Every DFD element must have a unique ID.
	5	Connecting the DFD elements with data flows must be possible.
	6	Each DFD element can be labeled with text.
	7	Each label can be edited.
System	1	Embed Draw.io.
	2	Must realize when a data flow crosses a trust boundary.
	3	Must recognize which elements a data flow connects and its direction.
	4	Should be a web-based application.
	5	Created DFDs can be exported in a meaningful format.
	6	Created tables can be exported in a meaningful format.
Threat Analysis	1	Automatically generate the STRIDE-per-Interaction overview table in the described format.
	2	Let the user tick and untick the checkboxes in the STRIDE columns of the overview table.
	3	In the interaction column, include the IDs of the two interacting elements. For example, ID-1 → ID-2
	4	User can input text after the element IDs to describe the corresponding interaction.
	5	For every data flow that has checked threats, generate a threat table in the defined format.
	6	Automatically fill in the threat ID, data flow ID and STRIDE Type in the threat table.
	7	Let the user input text into the threat, mitigation, and testing cells.
	8	Allow the user to create additional rows in the threat table.

4.2 Architectural Diagram

In Figure 4.1, the MVP requirements are being outlined. The high-level diagram of the system highlights the main components and illustrates how the application is constructed by following the flow through the process layer. It seems appropriate to encapsulate the system requirements as a bridge between modeling and threat analysis. Apart from the web interface, all subsequent processes should be viewed as actions that the user needs to complete or that the tool automates. Within the modeling process, the activity of creating a DFD is addressed. Once this task is finished, the tool generates an overview table, which the user then needs to complete. Concurrently, threat tables are generated based on the checkmarks from the overview table. These threat tables also need to be completed by the user, providing information about the threat, mitigation, and testing.

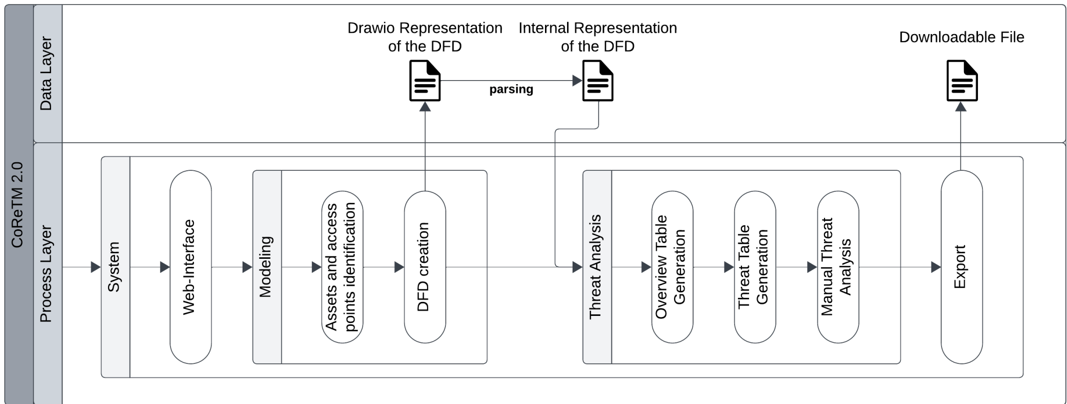


Figure 4.1: Architectural Diagram of the *CoReTM 2.0* MVP

Finally, a document is created and made available for the user to download. It should be mentioned that the data layer is intentionally kept as simple as possible. Even though the modeling process will generate a representation of the DFD according to Draw.io, this representation must be parsed to create the correct overview table. However, how this parsing process is being achieved is abstracted since it is considered too low-level.

4.3 Further Features and Changes

After presenting the MVP to the industrial partner, some changes and additional features were discussed, laying the groundwork for the second iteration of the development process. In this section, these additional features and changes are discussed.

The first change discussed was that the user should always be able to download the application's current state. This means that even unfinished work should be exportable to allow users to continue their work later. Although this functionality is already partially integrated into the MVP using local storage, it is valuable for the user to have the option to export their threat model at any point in the process. Additionally, the import functionality should be provided so that the user can import models and continue working on them. Another point raised by the industrial partner was that the data flow elements should have an ID attached. While Draw.io automatically assigns an ID to every element within the DFD, it is not visible to the user on screen. Since the textual description of the data flow does not have to be unique, a visible signifier was requested. A third feature discussed for the second development iteration is the ability for the user to add more rows to the threat tables manually. This functionality is needed as a specific interaction may have multiple threats of any STRIDE threat type. Currently, the MVP only creates one row for each STRIDE threat type present, but allowing the user to customize the dimension of the threat table after it is generated would solve this. Finally, the industrial partner requested a change in how the threat tables are built. A threat table was originally generated for each row in the overview table with at least one STRIDE threat present, following the STRIDE-per-Interaction methodology. However, the industrial partner asked to deviate from this and generate threat tables related to the trust boundaries within which threats

can occur. This results in fewer threat tables but more rows per threat table. This change results in a grouping that bundles all the threats present within a trust boundary without any loss of threat table rows.

Chapter 5

Design & Implementation

In this chapter, the design decisions, as well as the implementation of the *CoReTM 2.0* prototype, are discussed. Since the implementation of *CoReTM 2.0* is done in iterations and conducted with an industrial partner, the following section gives a general overview of the used technologies rather than a detailed implementation description of these elements used, and Section 5.2 covers the technical implementation of the MVP, which was discussed in the previous chapter. After each development iteration, the industrial partner is presented with the current state of the application, and feedback is gathered. This feedback and further inputs about changed requirements are then considered in the subsequent development cycle; information about the enhancements from the MVP to the second iteration can be found in Section 5.4. After the technical details of the MVP are explained, a demonstration of the MVP is provided, followed by the implementation details and changes from the second development iteration.

5.1 Technology Stack

The choice of stack and technologies is crucial for ensuring the application's performance, scalability, and maintainability. Given that *CoReTM 2.0* is a web application, choosing the correct programming language and framework is crucial.

The decision to use **TypeScript** as the primary programming language for the *CoReTM 2.0* prototype is underpinned by several key factors. TypeScript is a statically typed superset of JavaScript that enables catching type-related errors at compile time rather than at runtime [48]. This significantly reduces the likelihood of bugs and improves code quality. JavaScript remains the most widely used programming language, as highlighted by the 2024 StackOverflow survey, while TypeScript takes the 5th place [49]. Additionally, TypeScript offers superior tooling support, including autocompletion, refactoring, and intelligent code navigation, which boosts productivity and efficiency. The non-functional prototype of *CoReTM* was developed using JavaScript. For the development of *CoReTM 2.0*, the React framework was added to build a component-based user interface. The author of this thesis is already familiar with this framework. This familiarity reduces

the learning curve and accelerates the development process. React is used to build the user interface due to its component-based architecture, which makes it easier to create dynamic and interactive web applications [50].

As discussed in the architecture section, the modeling part of the MVP is implemented using **Draw.io**. This is achieved by embedding the Draw.io editor within an `Inline Frame element` (iFrame). How this is accomplished is discussed in Section 5.2. The steps for embedding and creating a custom shape library, which ensures users build DFDs with the correct set of symbols, are detailed in [51]. The stencils used for *CoReTM 2.0* are elements already present in other libraries of Draw.io. However, to differentiate between them, a tag has been manually added to each. This was accomplished by modifying the XML and adding a "type" attribute to each element, distinguishing the elements from one another. This facilitates the parsing of the diagram, which is covered later on. A notable project that aligns with this approach is ThreatFinder.ai [52]. While detailed information about this application is not provided here, ThreatFinder.ai is open source and incorporates a Draw.io embedding, allowing certain implementation parts to be adopted without reinventing the wheel [53].

Material UI (MUI) is utilized for styling the application [54]. MUI is a popular React component library that implements Google's Material Design principles, offering a set of pre-designed and customizable components. This not only accelerates development by reducing the need for custom CSS but also ensures a consistent and aesthetically pleasing user interface.

In summary, the combination of TypeScript, React, Draw.io, and Material UI forms a robust technology stack that supports the development of a dynamic, scalable, and maintainable web application. The Web Storage API is used since no persistency layer is present in this application. This allows the storage of necessary objects like strings and arrays within the browser [55].

5.2 Components

This section delves into the technical implementation details of the various components of the *CoReTM 2.0* MVP. Each component plays a specific role in the application's overall functionality, and their design and implementation are crucial for the project's success.

While *Views* and *Interfaces & Storage* give a superficial explanation of how the navigation through the app is structured, as well as how the internal objects are represented and stored, the other two subsections, *Modeling & Parsing* and *Tables*, offer a deeper insight into the business logic of the MVPs functionality. Covering the topics of how modeling is enabled, how changes within the DFD are registered, how the corresponding tables are calculated, and more.

5.2.1 Views

Views are the user-facing parts of the application, responsible for displaying data and interacting with the user. This subsection discusses the definition of different views within the *CoReTM 2.0* application, focusing on how they are structured and rendered using React.

Listing 5.1: Showcase different Links to Routes

```
// import statements
function Home() {
return (
    <ThemeProvider theme={theme}>
        <Grid container>
            <Container>
                // some typography elements
                <Stack>
                    <Link to={"/import"}>
                        <Button>
                            Import
                        </Button>
                    </Link>
                    <Link to={"/model"}>
                        <Button>
                            Create
                        </Button>
                    </Link>
                </Stack>
            </Container>
        </Grid>
    </ThemeProvider>
    );
}
export default Home;
```

When accessing the website, users are presented with a landing page where they can choose to create a new threat model or click on the import button. This can be seen in Listing 5.1. Although not part of the MVP requirements and therefore non-functional, the import button has already been designed as a stretch goal. To navigate to a specific route, users can click on the `<Button />` element wrapped within a `<Link />`, which eventually redirects them. This functionality is accomplished using react-router version 6.23.1. Despite the availability of later versions for this application, the used version is sufficient [56]. The implementation of this functionality is detailed below. In the main component `App.tsx`, the router is set up with three different routes, with the empty route (*i.e.*, the path `/`) serving as the landing page; how this is setup can be seen in Listing 5.2. Additionally, the landing page includes an error element `<NotFound />`, which is rendered if users attempt to access undefined routes. This prevents users from getting stuck in an invalid route. In the `Home.tsx` file, the `<Link />` elements can be observed that lead to the other routes. Furthermore, the `<Home />` component also includes some intentional omissions in styling attributes. However, it can be seen that it is wrapped within a `<ThemeProvider>` element imported from the MUI library that provides this functionality.

Listing 5.2: Definition of the Routes within the Application

```
// import statements
const router = createBrowserRouter([
    {
        path: "/",
        element: <Home />,
        errorElement: <NotFound />
    },
    {
        path: "/import",
        element: <Import />
    },
    {
        path: "/model",
        element: <Model />
    }
]);

function App() {
    return (
        <RouterProvider router={router} />
    );
}
export default App;
```

5.2.2 Interfaces and Storage

Interfaces establish contracts between various parts of an application, ensuring that components can communicate effectively and that the objects being passed through are based on a common blueprint. Additionally, the data must be explicitly stored so the user can export their threat model at the end of the process. This section covers the design and implementation of these interfaces, emphasizing the importance of clear and consistent communication between different modules and their storage methods. Two different interface types exist within the MVP of *CoReTM 2.0*.

Draw.io Interfaces

In the Draw.io Interfaces, the DFD's various elements are translated from an external representation to an internal one. This is achieved by the parsing algorithm, which is discussed in detail in the next section. Since the external representation contains unnecessary data, it's logical to convert them into a format that allows for better accessibility, differentiation between the different DFD elements, and concentration of the data by storing only the relevant information in a structured manner.

The central part of the Draw.io interface consists of three different interfaces: `IElement`, `ITrustBoundary`, `IDataFlow`. The `IElement` interface includes all DFD elements that are *1.* part of the *CoReTM 2.0* stencil and *2.* not of type `ITrustBoundary` or `IDataFlow`. Thus `IElements` can be *e.g.*, Data Stores, Interactors, Processes or Multi-Processes. The

interface is defined as per Listing 5.3. The other two interfaces are treated differently from the `IElements`. The main difference between an object of type `IElement` and one of type `ITrustBoundary` is that `IElements` can be within a (or multiple) `ITrustBoundary`, the interface of `ITrustBoundary` is depicted in Listing 5.4. Which `IElements` are connected by a `IDataFlow`, can be observed when taking a look at the `IDataFlow` interface 5.5. Each object that is of type `IDataFlow` must have a `sourceId` and a `targetId`; these IDs correspond to the `IElements` that are connected through the `IDataFlow`. This information is crucial to distinguish which `IDataFlows`, connecting the different `IElements` must be considered for the threat analysis and which are irrelevant. The reason for this will be evident when the parser algorithm is explained. Still, for now, it's sufficient to understand that those three categories exist and that each diagram constructed within the Draw.io embedding can be translated into an internal representation using these interfaces.

Listing 5.3: Definition of the IElement Interface

```
1  export interface IElement {
2      id: number
3      name: string
4      type: string
5      x1y1: { x1: number, y1: number }
6      x2y1: { x2: number, y1: number }
7      x1y2: { x1: number, y2: number }
8      x2y2: { x2: number, y2: number }
9      inTrustBoundary: Array<number>
10 }
```

Listing 5.4: Definition of the ITrustBoundary Interface

```
1  export interface ITrustBoundary{
2      id: number
3      name: string
4      type: string
5      x1y1: { x1: number, y1: number }
6      x2y1: { x2: number, y1: number }
7      x1y2: { x1: number, y2: number }
8      x2y2: { x2: number, y2: number }
9  }
```

Listing 5.5: Definition of the IDataFlow Interface

```
1  export interface IDataFlow{
2      id: number
3      name: string
4      type: string
5      sourceId: number
6      targetId: number
7  }
```

Within the Draw.io Interfaces, two more Interfaces exist: `IDiagram` and `ICrossingElements`. They serve the purpose of creating an accessible object that contains all the DFD elements as arrays (`IDiagram`) as per Listing 5.6 and contains all the necessary information to calculate the dimensions of the overview table, which must be derived (`ICrossingElements`). How this is implemented can be seen in Listing 5.7. This can be observed by having a look at their definition:

Listing 5.6: Definition of the IDiagram Interface

```
1  export interface IDiagram {
2      dataFlowsArray: Array<IDataFlow>,
3      elementsArray: Array<IElement>,
4      trustBoundariesArray: Array<ITrustBoundary>
5  }
```

Listing 5.7: Definition of the ICrossingElements Interface

```
1  export interface ICrossingElements {
2      dataflow: IDataFlow,
3      elements: {
4          sourceElement: IElement,
5          targetElement: IElement
6      }
7  }
```

TableRow Interfaces

As covered in Chapters 2 and 4, applying STRIDE-per-Interaction requires two different types of tables. The first table created after the user finishes the modeling phase is called the overview table. The number of rows is defined by the number of `IDataFlows` connecting two elements that are not within the same `ITrustBoundary`. This information is derived by parsing the given DFD and analyzing its structure. Therefore, an interface is created that contains all the information required to represent one row in the overview table. The implementation of this interface is visible in Listing 5.8. The attributes `type, dataflowId, dataflowName, interaction` are derived from the DFD. The user has to provide a description for each of these interactions and then decide which of the STRIDE threats is present in this interaction.

Listing 5.8: Definition of the IOverviewTableRow Interface

```
1  export interface IOverviewTableRow {
2      type: string,
3      dataflowId: number,
4      dataflowName: string,
5      interaction: string,
6      description: string,
7      threat: {
8          S: boolean,
9          T: boolean,
10         // R, I, D, E
11     }
12 }
```

The second type of table in STRIDE-per-Interaction is the threat table. These tables are created by analyzing the information in the overview table. Originally, a single threat table was generated for each row in the overview table. However, the industrial partner wanted to change this so that the threat tables do not correspond to individual rows but instead group together all the `IDataFlows` and their associated threats within the same `ITrustBoundary`. Although this change was not part of the MVP, it does not affect the construction of the `IThreatTableRow` interface, which is defined in Listing 5.9. The format for each row is already defined in the architecture chapter. A user needs to supply detailed information about each threat, including its STRIDE type, a mitigation strategy, and a way to validate the mitigation. The system automatically generates the other attributes for an object of type `IThreatTableRow`.

Listing 5.9: Definition of the IThreatTableRow Interface

```
1 export interface IThreatTableRow {
2     type: string,
3     threatId: string,
4     dataflowName: string,
5     strideType: string,
6     threat: string,
7     mitigation: string,
8     validation: string
9 }
```

Storage

Storing data effectively is crucial for ensuring the user's progress is maintained, and the threat model can be exported for further analysis or sharing. In the *CoReTM 2.0* MVP, local storage is utilized to handle data persistence and changes within the DFD and tables. This method offers a simple and dependable way to store data on the client's browser without needing a backend server. It aligns with the MVP's lightweight and self-contained design and enhances security by not depending on third-party applications or storage solutions.

Local storage in web applications refers to a browser's ability to store data locally on the user's device. This data is persistent across sessions, meaning that if a user closes their browser or navigates away from the application, their data will still be available when they return [55]. This persistence is essential for *CoReTM 2.0*, where users might spend considerable time constructing complex threat models and must ensure their work is not lost. In the *CoReTM 2.0* MVP, the data that must be persisted is *1)* the DFD data, *2)* the overview table, and finally *3)* the threat tables. This is done to allow users to come back and resume their work from the last saved state and to enable downloading upon completion of their work. Details about the implementation of saving to local storage, updating current states, and exporting the final result are explained thoroughly in Section 5.2.3.

To facilitate the export of the threat model, the data is structured in a way that can be easily converted to JSON (JavaScript Object Notation) format. JSON is a lightweight data-interchange format that is easy for humans to read and write and for machines to

parse and generate [57]. By structuring the objects according to the defined interfaces (`IDiagram`, `IElement`, `ITrustBoundary`, `IDataFlow`), the data can be exported seamlessly. The code snippet below shows how this is achieved 5.10.

Listing 5.10: Exporting Local State to JSON and Invoking Download

```
1  function downloadLocalStorageAsJSON() {
2      const localStorageData : any = {
3          "Diagram": JSON.parse(localStorage.getItem("DrawioMsg")!).xml,
4          "OverviewTable": localStorage.getItem("OverviewTable") || "[]",
5          "ThreatTables": localStorage.getItem("ThreatTables") || "[]"
6      }
7
8      const jsonString = JSON.stringify(localStorageData, null, 1);
9      const blob = new Blob([jsonString], { type: "application/json" });
10     const link = document.createElement("a");
11
12     link.href = URL.createObjectURL(blob);
13     link.download = `${projectName}.json`;
14     link.click();
15
16     URL.revokeObjectURL(link.href);
17 }
```

By utilizing local storage for persistence and structuring data for export in JSON format, the *CoReTM 2.0* MVP ensures that users can save their progress, retrieve it across sessions, and export their threat models for further use. This approach provides a robust and user-friendly way to manage data within the application.

5.2.3 Modeling and Parsing

One of the core functionalities that the MVP provides is that the user can model the system in the form of a DFD. This is depicted in Figure 4.1 as "Modeling". To achieve this, the Draw.io API is being embedded into *CoReTM 2.0*. This embedding is thematized in this subsection, and how the external representation is analyzed to build the internal representation using the interfaces introduced above. The embedding occurs in the `<DrawIO>` component. This component's two most important parts are the `useEffect` hook and the `iFrame` that is returned and rendered. Since the `useEffect` hook only runs once, after the initial render of the component, it is being used to initialize the `DrawioController` as well as the `TablesController`. The `DrawioController` is a class that takes three parameters in the constructor: *1)* a `CORSCommunicator` that handles the messages from the Draw.io embedding in both directions, *2)* the `LocalStorageModel` and lastly *3)* the project name the user provided. The constructor is depicted in the Listing 5.11.

Listing 5.11: DrawioController Constructor

```
1  constructor(drawio: CORSCommunicator, storage: LocalStorageModel, projectName:
       string) {
2          this.drawio = drawio
3          this.storage = storage
4          this.diagramAnalyser = new DiagramAnalyser();
5          this.projectName = projectName;
```

```
6         this.diagramExportPng = "";
7         this.drawio.receive(this.handleIncomingEvents.bind(this))
8     }
```

The DiagramAnalyser seen within the constructor is part of the parsing functionality and is covered later. As mentioned, the second and most important thing to be able to model the system is the iFrame that is rendered within the <DrawIO> component. This defines the interface that is visible to the user and lets them interact with it to create DFDs, as seen in Listing 5.12.

Listing 5.12: IFrame of Draw.io embedding

```
1 <iframe
2     ref={iframeRef}
3     width="100%"
4     height="700"
5     src="https://embed.diagrams.net/?embed=1&...&proto=json&configure=1&..."
6     style={{ border: "none" }}
7     title="draw.io"
8 />
```

The two most important parts of this iFrame are the ref and src attributes. The ref attribute is initialized with the useRef hook and a value of null. Still, it will eventually create a reference to the iFrame element, which is a mutable object that persists for the component's lifetime. Once the iFrame is mounted, the iframeRef.current is passed to the CORSCommunicator to allow it to communicate with the embedded iFrame, enabling functionalities like sending and receiving MessageEvents. This is crucial for updating the DFD while the user is editing it.

The src attribute defines the URL of the content to be embedded. The most important part of this URL is that the query parameters embed and configure are set to 1, and proto is set to JSON. This allows sending a custom configuration through the CORSCommunicator. For example, it will enable custom stencils to be used or the CSS properties of the embedding to be customized. This configuration can be found in the DrawioController and is triggered as soon as the CORSCommunicator receives a "configure" message. Furthermore, the proto attribute specifies that the protocol for communication with the editor is in JSON format [58].

In the MVP, an important feature is found in the DrawioController class. The private method handleIncomingEvents is responsible for handling inbound messages from the CORSCommunicator, as seen in Listing 5.13. This method is bound in the constructor, as shown in Listing 5.11, and is triggered every time a message is received. Depending on the type of event within the message, a different flow is initiated. This is achieved by a switch statement that acts as the control point, distributing the various events to different methods. Within the switch statement, we can see the "configure" event mentioned above and others. For example, the "autosave" event is always received when something in the diagram has changed, triggering an update of the currently saved model in the local storage. The "init" event is received when the iFrame is loaded. If the user has a stored model in their local storage, this model is loaded upon receiving this event. Otherwise, a blank diagram is loaded.

Listing 5.13: Incoming Events from Draw.io

```
private handleIncomingEvents(message: any) {
    if (message.data.length <= 0) {
        return
    }
    if (!this.isJsonString(message.data)) {
        return
    }
    const msg = JSON.parse(message.data);

    switch (msg.event) {
        case "autosave":
            this.autoSaveDiagram(msg);
            break;
        case "export":
            this.storeDiagram(msg);
            break;
        case "init":
            this.loadDrawio();
            break;
        case "configure":
            this.configureDrawio();
            break;
        default:
            console.error("Unknown event: ", msg.event);
    }
}
```

After the user completes the DFD, they can trigger the parsing process by clicking the "Analyse" button. The parsing, or `parseXml` method, is the only public method in the `DrawioController` class and can be triggered from outside the class – in this case, it's triggered from the `<DrawIO>` component where the "Analyse" button is defined.

This method returns two values: an array of objects with the type `ICrossingElement` as defined in Listing 5.7 and a boolean called `invalidDataflows`. The name of the boolean suggests that the parsing analyzes the DFD and checks for invalid behavior. Since a custom element library is provided, only elements from this library are expected to be used in the DFD. Suppose the user decides to use other elements. In that case, a warning will be displayed, informing the user that any elements not part of the CoReTM stencil will be ignored in the analysis. This doesn't prevent the user from using these elements, but it's important to note that the resulting overview table may not have the expected dimensions.

Additionally, the `invalidDataflows` variable indicates if there are `IDataFlows` within the DFD that do not have a source or target point, or both. If any `IDataFlows` are found in this state, an alert is displayed, preventing the user from continuing with the threat analysis process until all the `IDataFlows` have a valid source and destination.

When the `parseXml` method is called by clicking the button, the actual work takes place within the `DiagramAnalyser` class. However, it is called through the `DrawioController`, which holds an instance of `DiagramAnalyser`. In the first approach, parsing occurred with every change of the diagram, eliminating the need for the user to trigger it manually.

Nevertheless, this approach led to unnecessary time complexity. The entire diagram had to be parsed for every small change, and the algorithm always had to compare the previous state with the updated state, resulting in extra computational overload. This became evident when after deleting an element from an existing DFD, it was still present in the internal representation. To reduce the computational overhead from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$, it was decided to parse the diagram only once, once the user signifies that they have finished their modeling by clicking the button.

Shifting the focus on the other variable returned by the parsing, which is `crossingElements`. Assuming that all the `IDataFlows` in the DFD have a source and target, and the other elements used are from the CoReTM stencil, what should the actual output be? After some consideration, the conclusion was that it should be a list of the `IDataFlow` objects present in the DFD that connect two different `IElements` which are not within the same `ITrustBoundary`; this is represented by the Listing 5.7. For this purpose, all `IElement` objects have an attribute `inTrustBoundary`, which is an array of numbers representing the IDs of the `ITrustBoundary` objects the `IElement` object lies within. If there are two different `IElement` objects connected through an `IDataFlow`, their `inTrustBoundary` arrays should contain at least one ID that is not in common. However, data engineering must occur upfront before identifying these elements.

Listing 5.14: Parsing invocation within the DrawioController

```
1  parseXml() : {crossingElements: ICrossingElements[], invalidDataflows: boolean} {
2      const xmlDataString : string | null = this.storage.read("DrawioMsg");
3      const parsed = JSON.parse(xmlDataString!);
4      const xml = parsed.xml;
5
6      let xmlDoc : XMLDocument;
7
8      if (xmlDataString) {
9          const parser = new DOMParser();
10         try {
11             xmlDoc = parser.parseFromString(xml, "text/xml");
12         }
13         catch (e) {
14             console.log(e);
15         }
16     }
17     const {crossingElements, invalidDataflows} =
           this.diagramAnalyser.parseDifferentDfdElementsFromXml(xmlDoc!);
18     if (crossingElements.length > 0) {
19         this.exportDiagram()
20     }
21
22     return {
23         crossingElements: crossingElements,
24         invalidDataflows: invalidDataflows
25     }
26 }
```

The `parseXml` method passes an `XMLDocument` to the `DiagramAnalyser`, as shown on line 17 in Listing 5.14. This `XMLDocument` contains much information; we are specifically interested in the diagram tag, particularly all the children of the diagram tag called mx-

Cells. These mxCells are extracted using `getElementsByTagName("mxCell")` within the `parseDifferentDfdElementsFromXml` method in the `DiagramAnalyser`; this is demonstrated in the Listing 5.15. Each mxCell represents an element within the DFD and contains essential information such as ID, value (editable textual description), type, and coordinates. If the mxCell is of type "Dataflow", it also includes the source and destination IDs of the connected elements. Some implementation details in the `parseDifferentDfdElementsFromXml` method are replaced with comments because they would rather confuse the reader and are not especially important for understanding the following parsing functionality. After retrieving all the mxCells present in the DFD, the instance variables of the `DiagramAnalyser` class are reset. This is important to avoid any data from earlier parsings that could interfere with and distort the final result. Looping over all the mxCells, the invalid ones are filtered out, and the valid ones are navigated into the correct array of the `diagramElements` instance variable after being translated to the corresponding interface type on lines 15-17. The `diagramElements` variable is of type `IDiagram` as defined in Listing 5.6. When the loop finishes, the `diagramElements` instance variable contains all DFD elements in the current model and separates them into different arrays. Afterward, on lines 20-21, a loop over all the `IElements` of the DFD is made to populate the `inTrustBoundary` attribute. Finally, all DFD elements connected with another one and cross a trust boundary are returned, along with all invalid data flows, meaning those that lack a source, target, or both.

Listing 5.15: Parsing the XMLDocument within the DiagramAnalyser Class

```
1  parseDifferentDfdElementsFromXml(xmlDoc: XMLDocument) : {crossingElements:
       ICrossingElements[], invalidDataflows: boolean} {
2      const mxCells = xmlDoc.getElementsByTagName("mxCell");
3
4      this.diagramElements = {
5          dataFlowsArray: new Array<IDataFlow>(),
6          elementsArray: new Array<IElement>(),
7          trustBoundariesArray: new Array<ITrustBoundary>()
8      };
9      this.elementsCrossingTrustBoundaries = new Array<ICrossingElements>();
10     this.notAllowedElements = [];
11     this.dataflowsWithoutSourceOrTarget = [];
12
13     Array.from(mxCells).forEach(cell => {
14         // filter out invalid elements
15         const elementToAdd = this.createElementToAdd(cell, geometryElement, type);
16         if (elementToAdd) {
17             this.navigateElementToCorrectArray(elementToAdd, type);
18         }
19     });
20     this.diagramElements.elementsArray.forEach(element => {
21         this.addInTrustBoundaryAttributeToDfdElement(element);
22     })
23     this.findDataflowsCrossingTrustBoundary();
24     if (this.notAllowedElements.length > 0) {
25         // raise an alert
26     }
27     return {
28         crossingElements: this.elementsCrossingTrustBoundaries,
29         invalidDataflows: this.dataflowsWithoutSourceOrTarget.length > 0
30     }
```

Two-dimensional calculus is needed to determine whether or not two connected elements lie within the same trust boundary. In Listing 5.16 the coordinates of each element are compared to the coordinates of each `ITrustBoundary` that is present within the DFD. Through this, it is distinguished if the area of the `IElement` lies within the area of the respective `ITrustBoundary` object or not.

Listing 5.16: 2D Calculation to determine if Element is in Trust Boundary

```
1 private calculateIfElementInTrustBoundary(element: any, trustBoundary: any):
      boolean {
2     return
3         element.x1y1.x1 >= trustBoundary.x1y1.x1 && element.x1y1.y1 >=
              trustBoundary.x1y1.y1
4         && element.x2y1.x2 <= trustBoundary.x2y1.x2 && element.x2y1.y1 >=
              trustBoundary.x2y1.y1
5         && element.x2y2.x2 <= trustBoundary.x2y2.x2 && element.x2y2.y2 <=
              trustBoundary.x2y2.y2
6         && element.x1y2.x1 >= trustBoundary.x1y2.x1 && element.x1y2.y2 <=
              trustBoundary.x1y2.y2;
7 }
```

This may seem like a complicated statement, but it's just comparing the four corner coordinates defined for each `IElement` with the four corner coordinates defined for each `ITrustBoundary`. If the coordinates lie within the boundary, the statement returns true. Refer to Figure 5.1 to better understand this method.

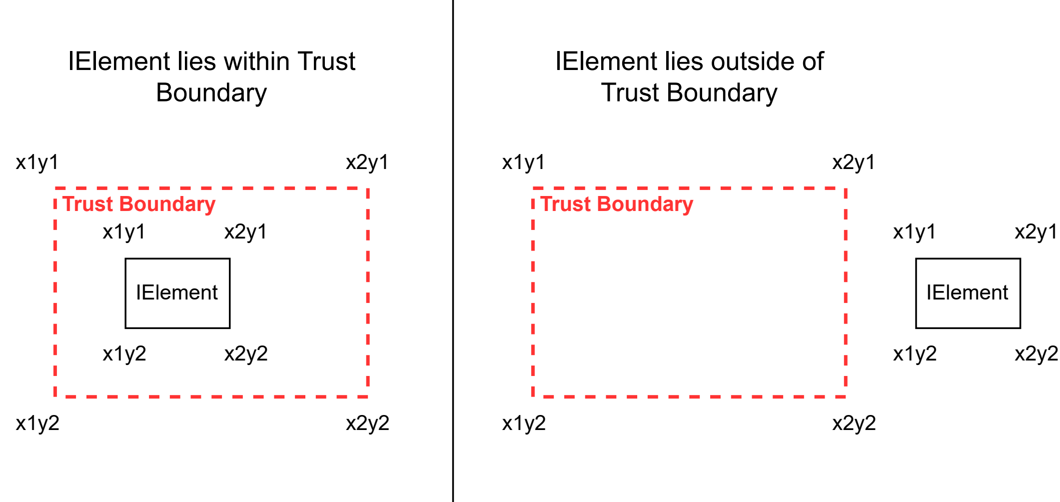


Figure 5.1: Visualization of Element's Calculation Within Trust Boundary

5.2.4 Tables

Once the DFD is parsed and the elements connected by a `IDataFlow` but crossing a trust boundary are calculated, it's time to create the overview table. There is a component called `<OverviewTable>` that takes the calculated `crossingElements` and a callback function `onSave`. This function is responsible for informing the parent component,

<DrawIO>, when the save button is clicked within the <OverviewTable> component. The system must know when the user has finished editing the information in the table. The information provided by the overview table is needed to generate the threat tables, so it is essential to establish when the user has finished editing.

Listing 5.17 has two different state variables. The first holds the overview table's current state, while the second waits for the user to click the save button to finish editing the overview table. After mounting the component, the `useEffect` hook is run. This hook initializes the `tableData` object, which contains all the information needed within the overview table. After looping over all elements within the `crossingElements` array, the `tableData` object is set as the current state of the `overviewTable` using the `setOverviewTable` function.

Listing 5.17: Function Definition of the OverviewTable Component

```
1  export default function OverviewTable({ crossingElements, onSave }: {
       crossingElements: ICrossingElements[], onSave: (data: IOverviewTableRow[]) =>
       void }) {
2      const [overviewTable, setOverviewTable] = useState<IOverviewTableRow[]>([]);
3      const [saveClicked, setSaveClicked] = useState(false);
4
5
6      useEffect(() => {
7          const tableData = crossingElements.map((element) => ({
8              type: "OverviewRow",
9              dataflowId: element.dataflow.id,
10             dataflowName: element.dataflow.name,
11             interaction: `${element.elements.sourceElement.name} ->
                   ${element.elements.targetElement.name}`,
12             description: "",
13             threat: {
14                 S: false,
15                 T: false,
16                 R: false,
17                 I: false,
18                 D: false,
19                 E: false
20             }
21         }));
22         setOverviewTable(tableData);
23     }, [crossingElements]);
24
25     // more functions that handle changes within the table and update the state
26
27     return (
28         // HTML of the table to be rendered & a button to save the data within the
               table
29     )
30 }
```

The visible table contains columns as outlined in Chapter 4. Users must describe each interaction by entering text in the description column, which is presented as a <TextField />. Additionally, users must indicate the presence of any STRIDE threat within an interaction by checking the corresponding <Checkbox />. By default, all the <Checkbox

/> elements are set to false, aligning with the initialization of the overviewTable variable after the useEffect hook has been executed. When the save button is clicked, the current state of the overview table is sent to the parent component. From the parent component, it is passed to the TablesController for parsing. The overview table needs to be parsed because the threat tables generated later are based on the selected STRIDE threats. In the MVP, a threat table is generated for each row in the overview table. However, this requirement changed after the MVP was presented to the industrial partner. The desired changes will be discussed later. When the overview table is parsed, it is saved in the local storage, and the structure of the threat tables is defined. This means that even if the user changes the text in the <TextField /> or the state of the <Checkbox />, it will no longer affect the threat tables or the saved version in the local storage. This issue needs to be addressed in the second iteration as well.

The threat tables instantiated by the TablesController are handled similarly to the overview table. The main difference is that rather than one table, it can be multiple. This complicates the updating of these tables. Therefore, instead of constantly looping over all tables and all cells, a LookupMap is generated, where each row is being stored, and the cells can be accessed and updated much more efficiently. This occurs when the component is mounted in the useEffect hook, as per Listing 5.18. Since the function receives the threat tables as input in the correct dimensions and the proper amount of threat tables, this can be done before the user even changes anything.

Listing 5.18: Function Definition of the ThreatTables Component

```
export default function ThreatTables({ threatTables, onSave } : { threatTables: IThreatTableRow[][], onSave: (data: IThreatTableRow[][]) => void }) {
    const [threatTable, setThreatTable] = useState<IThreatTableRow[][]>(threatTables);
    const [saveClicked, setSaveClicked] = useState(false);
    const lookupMapRef = useRef<Record<string, IThreatTableRow>>({});

    useEffect(() => {
        const generateLookupMap = (table: IThreatTableRow[][]) => {
            const map: Record<string, IThreatTableRow> = {};
            table.forEach((rows, index) => {
                rows.forEach(row => {
                    map[`${index}-${row.threatId}`] = row;
                });
            });
            return map;
        };
        lookupMapRef.current = generateLookupMap(threatTable);
    },);

    // more functions that handle changes within the tables and update the state

    return (
        // HTML of the tables to be rendered & a button to save the data
    )
}
```

After filling in the “Threat,” “Mitigation,” and “Validation” cells, the threat tables can be saved by clicking on the save button. This will store the threat tables in the local storage,

and a download button will appear. Clicking on this download button will conclude the threat modeling process and allow exporting the generated data in JSON format, as shown in Listing 5.10.

5.3 Demonstration

In this demonstration, the reader is visually guided through the application. This section shows the discussed views and how the components are rendered on screen. The data and diagram used in the following Figures are fictitious and serve as placeholders to showcase the application's MVP.

Figure 5.2 shows the landing page. It is held as minimal as possible so the user perceives two possible actions. Namely, clicking on either the "import" or "create" button. Since the "import" button is not implemented as mentioned, the following figures show the user's flow when the "create" button is clicked.

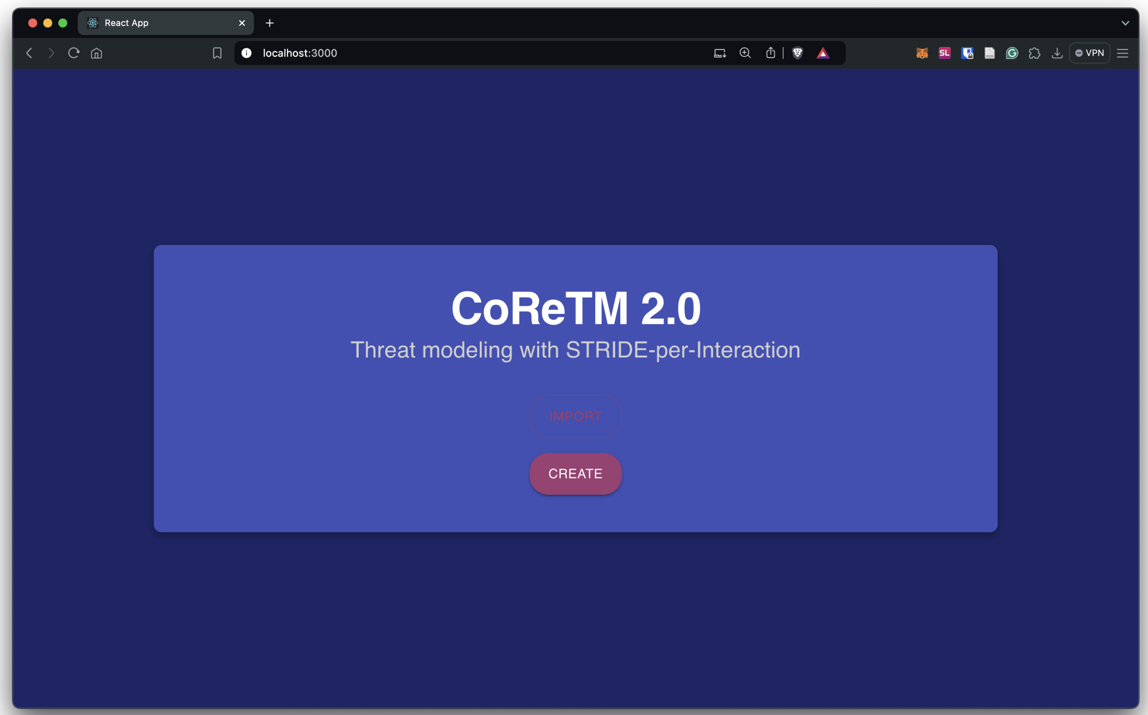


Figure 5.2: Landing Page *CoReTM 2.0* MVP

At first, the user must provide a project name, as seen in Figure 5.3. This is not a crucial functionality but should help the user give a meaningful name, which will finally be used for the export functionality. The name of the downloaded file defaults to the project name.

As soon as the project name is submitted, the drawio embedding is rendered. Figure 5.4 shows the `iFrame` of the Draw.io embedding, where the user can create the DFD using

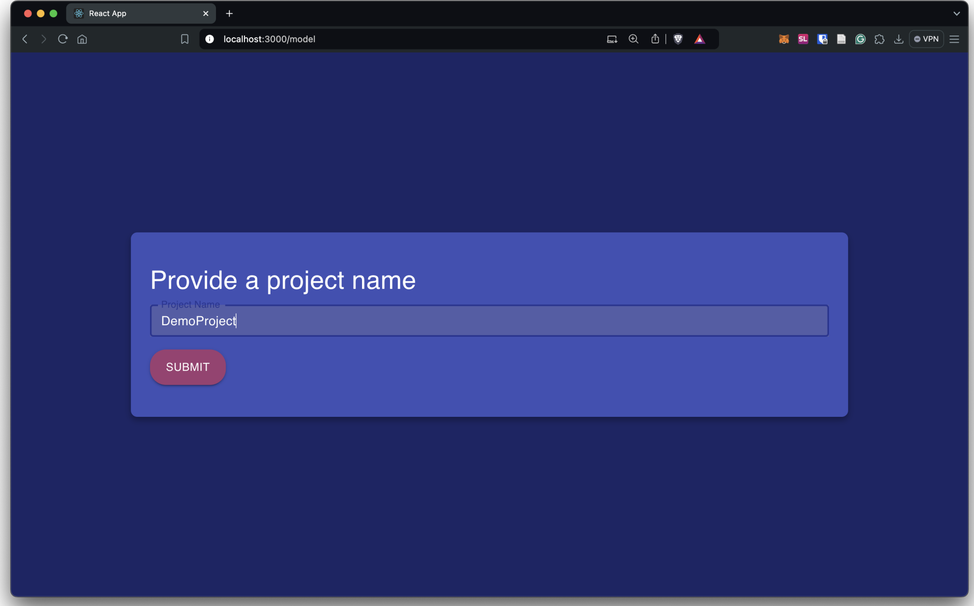


Figure 5.3: Define Project Name

the custom stencil. When the diagram is finished, the user is supposed to click on the "Analyse" button on the bottom right, which will trigger the parsing of the diagram.

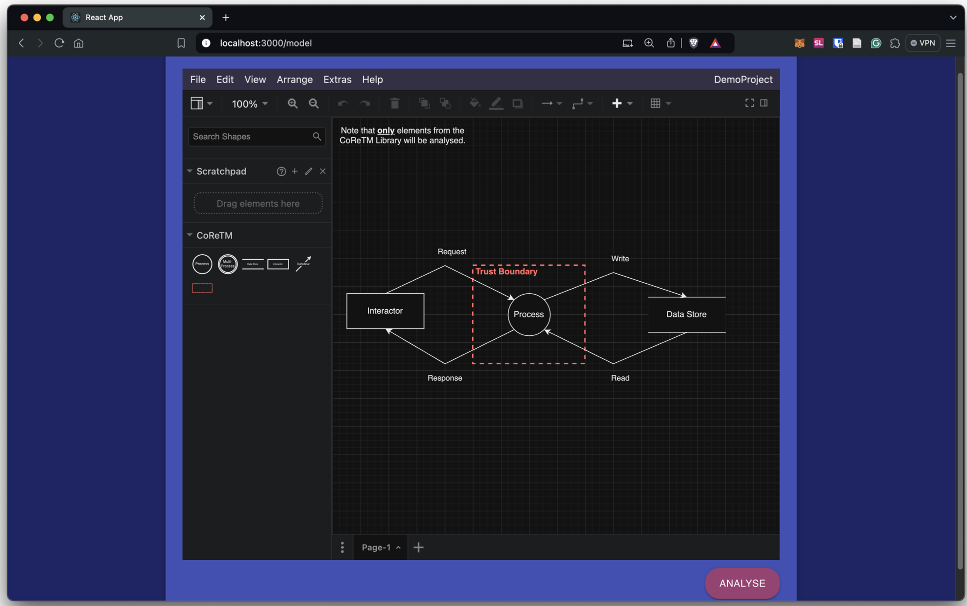


Figure 5.4: Drawio Embedding

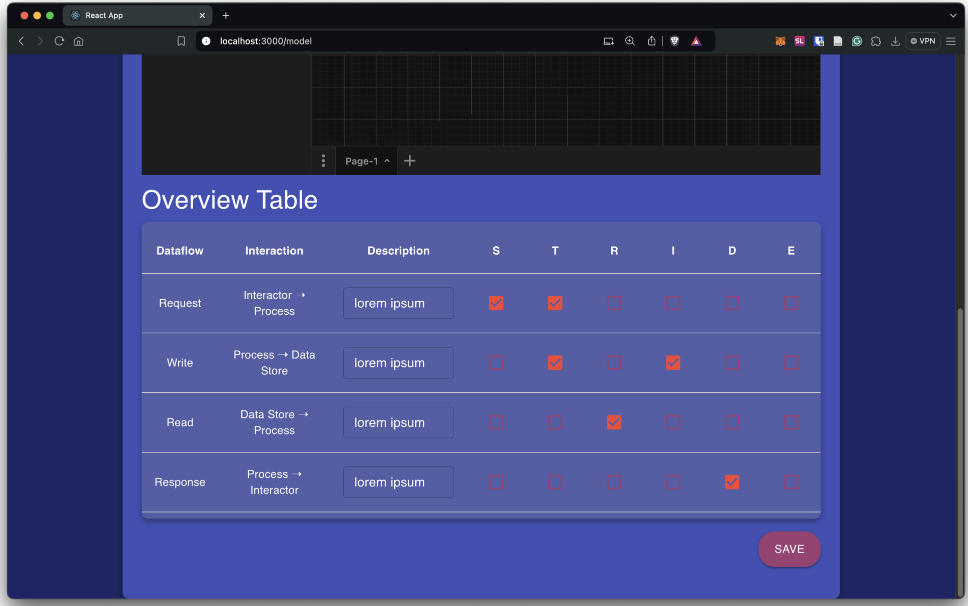


Figure 5.5: Overview Table

The parsing algorithm identifies all the data flows that connect two different DFD elements and cross a trust boundary. Figure 5.5 shows the overview table generated as a result of the analysis of the DFD. The user must describe each interaction and check the corresponding STRIDE threats by clicking the checkboxes. When all the fields are filled out and at least one checkbox per row is activated, the user can save the overview table and head on to the final step in the STRIDE-per-Interaction threat modeling process.

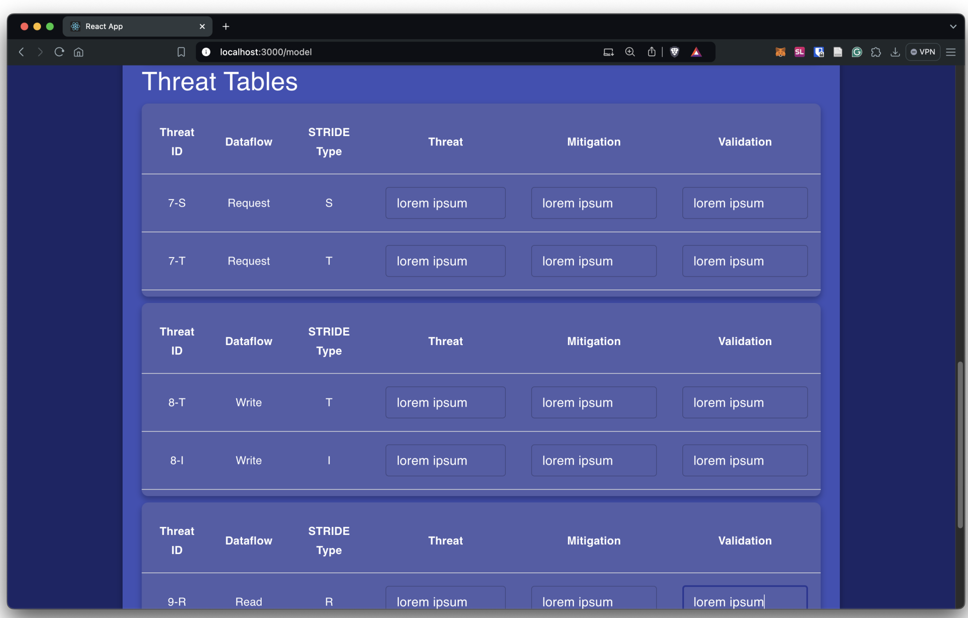


Figure 5.6: Threat Tables

By parsing the overview tables' checkboxes, the threat tables are generated. They can be partially seen in Figure 5.6. Like the overview table, the user must provide information by typing it into the text boxes. The application hinders the user from submitting any table (overview or threat tables) when some fields are not edited.

Finally, when all the information in the threat tables is complete, the user can click the "Download" button. Now, the user can decide where the file should be stored, as seen in Figure 5.7. This step completes the walk-through of the *CoReTM 2.0* MVP.

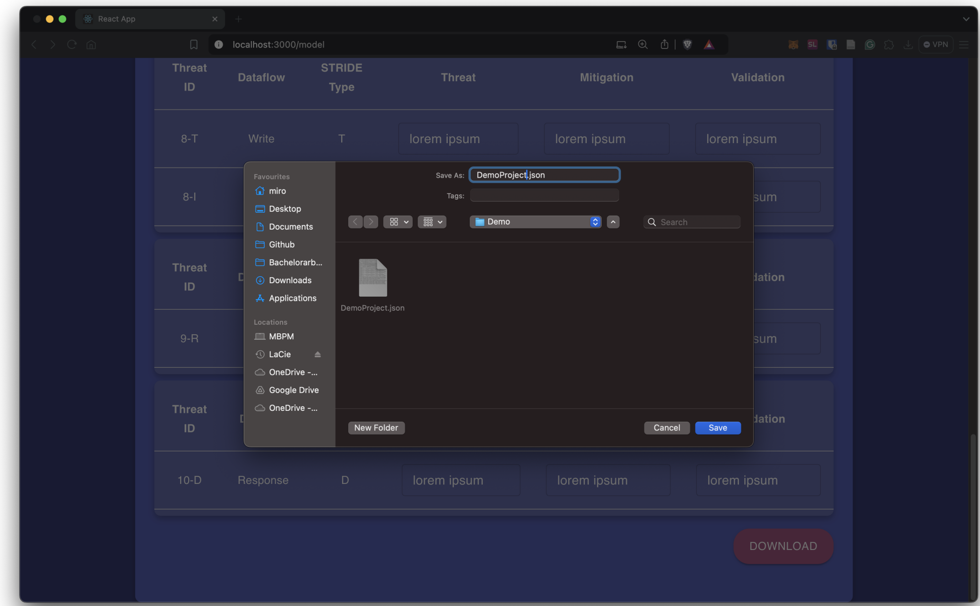


Figure 5.7: Download Threat Model

5.4 Second Development Iteration

This section discusses and explains the changes and advancements of the second development iteration. This iteration took place after presenting the MVP to the industrial partner, and additional features and changing requirements were gathered. These can be found in Section 4.3. Several minor changes and additions were made to the application in this iteration, but two major ones occurred. These major changes involved creating the import functionality and refactoring the overview and threat tables. As a result, the author of this paper chose to examine these major changes thoroughly and only briefly address the minor ones.

5.4.1 Minor Changes

One minor change that was not part of the feedback from the industrial partner but is considered to improve the user experience (UX) is that when the user finishes creating

their DFD and clicks the “Next” button, an image is rendered instead of the drawio editor. This ensures that when the overview table is shown, the user cannot change things in the DFD that would invalidate or impact the overview table. This could also be implemented dynamically, but it would imply that the DFD would have to be parsed for every change within the diagram, which was decided against due to the computational overhead. To achieve this, an `export event` has to be sent via the `CORSCommunicator`. The response to this is a Base64 encoded SVG, which can be rendered within an `<img>` HTML tag. The result of this adjustment can be seen in Figure 5.8.

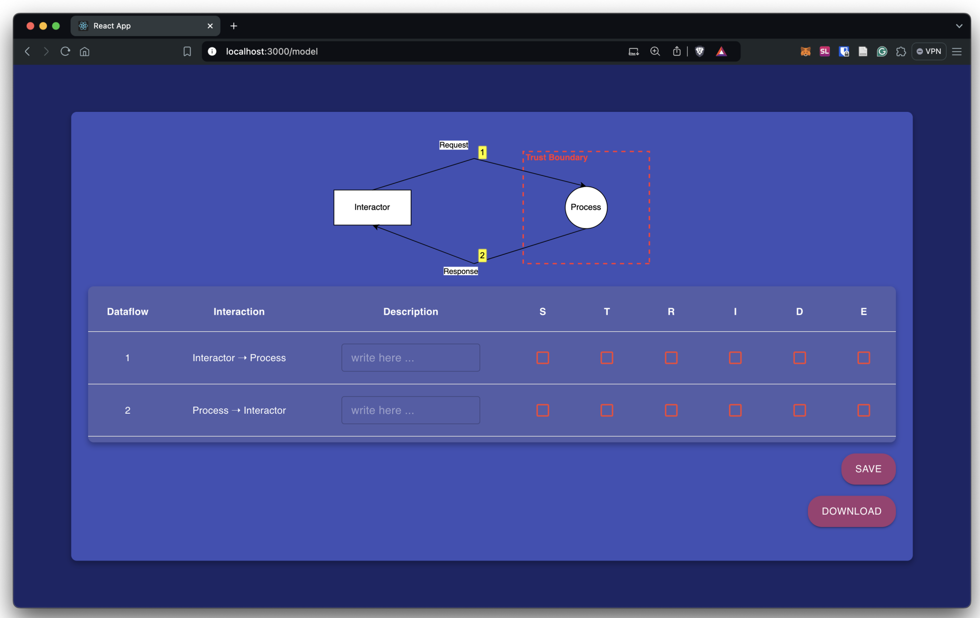


Figure 5.8: Rendering SVG after DFD completion

A further improvement to the UX has been implemented, allowing the user to download the current state of the threat modeling process at any point. In the initial version, the download feature was only available after the completion of the threat modeling process, when the “Download” button would appear. This was achieved through conditional rendering. The conditional rendering has now been updated, so the button will always be visible once the user provides a project name. This enhancement increases flexibility within the application, as users no longer need to finish the entire process to export it. However, another functionality is required to fully benefit from this feature – the ability to import an existing model. This will be addressed in the Section 5.4.2.

In the MVP feedback session, a requested feature was to visually distinguish data flows within the DFD with unique tags. Initially, it was assumed that users could label data flows, but the labels are not unique or mandatory. The default label is “Dataflow,” which remains unless edited. Additionally, multiple data flows could have the same label. To address this, the custom stencil’s metadata has been modified to enumerate data flows automatically. This enumeration is done dynamically, based on the data flows’ creation time, with older data flows having lower numbers and newer ones having higher numbers.

This improvement also enhances the UX by making it easier to find the referenced data flows in the overview table when reviewing DFD. Additionally, identifying a specific data flow is now unambiguous, as each data flow is enumerated, ensuring uniqueness. This behavior can be observed in the Figure 5.8.

Finally, for a last minor change, the `<NotFound>` component was created. This component is rendered when a user tries to access a path not defined by the router. The router can be seen in Listing 5.2. The component used to display a simple text with a button, without utilizing the MUI library. This was changed to align with the overall application design.

5.4.2 Major Changes

Two significant changes and refactorings were done in the second development cycle. One of these changes involves an entirely new addition to the code base and requires thorough coverage. The second change relates to modifications in how tables are generated and the functionality they provide. Since this deviates from the existing documentation in this chapter, it is important to mention it for completeness.

Import

The import functionality is being implemented by creating the `ImportController` class and the corresponding `Import` user interface (UI) react component. This allows users to upload a JSON file containing project data from earlier threat modeling sessions using *CoReTM 2.0*. This data is then parsed and stored in the browser's local storage. The `ImportController` class handles the core logic for parsing and validating the imported file content. It ensures that the JSON data is correctly formatted. On the other hand, the `Import` component provides the user interface for uploading and importing the file. The structure of the `ImportController` can be seen in Listing 5.19. The class contains just one method: `parseFile`. This method takes a `fileContent` string as input and returns a boolean value upon completion. It processes the content of the imported file step by step using multiple `if-statements` and writes the content to the local storage. The method also includes error handling; for example, if the imported diagram data is not in the correct XML format, an alert is displayed to the user. If any other error occurs during the method execution, the import process fails, and an error is logged into the console. Users can view the error using the browser's built-in developer tools.

If the user provides a valid JSON file with an existing model, all the contents are stored in the local storage and are loaded in the modeling view. It's important to note that the imported model doesn't need to be complete. Since the user can download the model they are creating at any point during the modeling process, the import functionality must adapt to this by supporting the import of partial models. This means that it's possible to, for example, import only an existing DFD without overview or threat tables. This provides the user with maximum flexibility.

Listing 5.19: ImportController Class Definition

```
export default class ImportController {
    parseFile(fileContent: string): { success: boolean } {
        try {
            const parsedData = JSON.parse(fileContent);
            if (!parsedData) {
                alert("Failed to parse the file. Please upload a valid model.");
                return { success: false };
            }

            localStorage.clear();

            if (parsedData.ProjectName) {
                localStorage.setItem("ProjectName", parsedData.ProjectName);
            }

            if (parsedData.Diagram) {
                const xmlDoc : XMLDocument = new
                    DOMParser().parseFromString(parsedData.Diagram, "text/xml")
                const parseError = xmlDoc.getElementsByTagName("parsererror");
                if (parseError.length > 0) {
                    alert("Failed to parse the Diagram XML. Please upload a valid
                        model.");
                    return { success: false };
                }
                localStorage.setItem("DrawioMsg", JSON.stringify({ xml:
                    parsedData.Diagram }));
            }

            if (parsedData.OverviewTable && parsedData.OverviewTable !== "[]") {
                const overviewTable = JSON.parse(parsedData.OverviewTable);
                localStorage.setItem("OverviewTable",
                    JSON.stringify(overviewTable));
            }

            if (parsedData.ThreatTables && parsedData.ThreatTables !== "[]") {
                const threatTables = JSON.parse(parsedData.ThreatTables);
                localStorage.setItem("ThreatTables", JSON.stringify(threatTables));
            }
            return { success: true };

        } catch (error) {
            console.error("Error parsing file:", error);
            return { success: false };
        }
    }
}
```

The ability to import had a significant impact on the components responsible for the rest of the threat modeling process. The system now needs to recognize if an existing model was imported and adjust accordingly. If a complete model is imported and the user doesn't change anything, the imported model should be displayed. However, if the user makes changes to, for example, the DFD after importing a model, it could affect the overview table and the threat tables. This issue was resolved by introducing state variables in the `<DrawIO>` component and the `DrawioController` class. These variables

provide information about whether something was imported and if changes were made after the import. This behavior allows for controlling the logic flow to react to changes, such as creating a blank overview table when the DFD changes or simply displaying the imported content. The behavior described can be observed, for example, in Listing 5.20. First, it checks if an overview table has been imported and whether the DFD did not change after being imported. If this is the case, the imported table can be displayed. Otherwise, if the overview table was not imported or if the diagram changed after the import, the diagram must be analyzed, and a blank overview table must be displayed.

Listing 5.20: Logic for Handling the Next Button Click

```
1  function handleClickNextButton() {
2      const importedOverviewTable = localStorage.getItem("OverviewTable");
3  
4      if (!drawioController!.getChangedAfterImported() && importedOverviewTable) {
5          const parsedOverviewTable = JSON.parse(importedOverviewTable)
6          tablesController!.createOverviewTableFromImport(parsedOverviewTable);
7          setOverviewTableImported(true);
8          setOverviewTable(tablesController!.getOverviewTable());
9          drawioController!.exportDiagram();
10         setShowOverviewTable(true);
11     } else {
12         const {crossingElements, invalidDataflows} = drawioController!.parseXml();
13         if (crossingElements.length > 0) {
14             if (!invalidDataflows) {
15                 tablesController!.createOverviewTableFromDrawio(crossingElements);
16                 setOverviewTable(tablesController!.getOverviewTable());
17                 drawioController!.exportDiagram();
18                 setShowOverviewTable(true);
19             }
20         } else {
21             alert("There are no dataflows crossing a trust boundary. Therefore
                   STRIDE-per-Interaction cannot be applied.");
22         }
23     }
24 }
```

Table Generation

In the second iteration of development, there were significant changes to the strategy of how the threat and overview tables are generated, along with adding some new functionalities. Following feedback on the MVP, the industrial partner requested changes to these tables as detailed in Section 4.3. Specifically, they asked for the following modifications: *1)* The threat tables should be dependent on the trust boundaries rather than the rows within the overview table, and *2)* the threat tables should allow for the addition of more rows, for example, when multiple threats of the same STRIDE-type exist within a specific interaction.

The import functionality resulted in certain architectural changes. Specifically, the `TablesController` is now responsible for generating different types of tables, which are then passed down to the UI components (`<OverviewTable>` and `<ThreatTables>`). In

the MVP, the overview table was directly passed down to the `<ThreatTables>`. The threat tables were then generated within the UI component. Additionally, after the user submits the overview table by clicking on the "Save" button, the overview table becomes disabled, meaning it can no longer be edited. This is important to ensure that the information and dimensions of the threat tables remain correct after being generated upon submission of the overview table.

Importing threat tables requires some special considerations. For instance, if a user imports an existing model and makes no changes in the DFD but selects some new checkboxes (or unselects) within the imported overview table, the normal behavior would be to generate blank threat tables. As a result, the user would need to provide all the threat descriptions, mitigation strategies, and validation mechanisms again, even for threats that did not change at all. However, the author found this behavior undesirable because it forces the user to repeat already done work. Therefore, a mechanism was implemented to check if some threats already existed and are still present even though the overview table changed. When this is the case, the information from the imported threat tables is still displayed. Additionally, newly checked threats in the overview table will appear as blank rows within the threat tables. This way, the user does not have to worry about losing progress when making minor changes to the overview table. This is implemented within the `TablesController` and can be seen in Listing 5.21.

Listing 5.21: Update generated Threat Tables from Import Data

```
1  public updateThreatTable(importedThreatTable: IThreatTableRow[][]) {
2      this.generateThreatTables()
3      importedThreatTable.forEach((table, tableIndex) => {
4          table.forEach((row) => {
5              const existingTable = this.threatTables[tableIndex];
6
7              if (existingTable) {
8                  const existingRow = existingTable.find(existingRow =>
                       existingRow.threatId === row.threatId);
9
10                 if (existingRow) {
11                     Object.assign(existingRow, row);
12                 } else {
13                     existingTable.push(row);
14                 }
15             } else {
16                 this.threatTables[tableIndex] = [row];
17             }
18         });
19     });
20 }
```

Chapter 6

Evaluation

The evaluation of *CoReTM 2.0* took place with the industrial partner after completing the second development iteration. A "User-Centered Evaluation" methodology [59] was chosen to assess the software. During a remote meeting, the industry partner ran the software on his machine and shared his screen. This meeting was recorded, and the evaluation is based on the analysis of this recording. He was instructed to use the application and attempt to create a threat model himself. The purpose was to observe his behavior and assess how intuitive and effective the application was. He was also asked to verbalize his thought process (*i.e.*, think-aloud) as he interacted with the software, giving insight into his cognitive processes. The partner's task was to create a new threat model using the STRIDE-per-Interaction methodology and download it at the end. If he encountered difficulties, the evaluator provided hints to prevent any deadlocks.

During the process, the industrial partner provided valuable suggestions for improvements based on his experience as a software architect. He clearly understood what constitutes a good application in both UI and UX. Subsequently, the partner was asked to provide feedback through a questionnaire. This questionnaire allowed the author to gather specific feedback. It allowed the partner to elaborate on the positive and negative aspects of using the software, leading to the discovery of potential areas for future work and possible improvements. The findings of this process are elaborated on in Section 6.2.

6.1 Questionnaire

The questionnaire assesses various aspects of the application, including functionality, completeness, usability, user interface, and integration/compatibility. By analyzing each of these aspects, the author aims to determine how well the application fulfills its intended purpose and performs in real-world scenarios. Additionally, organizing the critique based on these aspects helps the author identify areas for improvement in future work.

The **Functionality** dimension evaluates whether the application fulfills the given requirements and aligns with the original specifications. It also assesses whether the application produces consistent results compared to the manual application of the STRIDE-per-

Interaction method, ensuring that the tool's output is reliable and accurate. This is being gathered through the following questions.

Q1) Are there discrepancies between the originally defined requirements and the actual implementation of the application?

Q2) Does the application produce the same results as the manual application of the STRIDE-per-Interaction method?

Completeness examines whether the STRIDE-per-Interaction methodology is fully supported by the application, ensuring that all necessary features are implemented. It also identifies any gaps in functionality, such as missing features that were initially planned but not implemented, which could impact the overall effectiveness of the application. For that, the following two questions were posed.

Q3) Is the STRIDE-per-Interaction methodology fully supported by the application?

Q4) Are there any functionalities that were not implemented or are missing?

Another dimension is **Usability**. It focuses on how easy and intuitive the application is for users. It includes an assessment using the SUS [60] and evaluates whether any steps in the threat modeling process are unclear or confusing. Additionally, it considers whether the application enhances the efficiency of the threat modeling process, reducing the time and effort required compared to manual methods.

Q5) System Usability Scale

Q6) Were there any steps in the threat modeling process that were unclear or confusing?

Q7) Does the application speed up the threat modeling process compared to manual execution?

The **UI** dimension assesses the design and structure of the application's interface, ensuring it is well-organized and user-friendly. It also evaluates the effectiveness of feedback mechanisms, such as notifications for task completion or errors. It determines if the application provides adequate support when users make mistakes, guiding them effectively toward the correct input. This dimension is investigated through the following three questions.

Q8) Is the interface well-designed and well-structured?

Q9) Does the application provide clear feedback during use, *e.g.*, when a task is completed, or an error occurs?

Q10) Does the application support the user in case of incorrect input? If so, is the assistance sufficient and informative?

Finally, **Integration and Compatibility** examines how well the application fits into existing organizational processes and tools. It also assesses whether the application performs smoothly across different devices and browsers, ensuring broad accessibility and usability in various technical environments.

Q11) Does the application allow seamless integration into existing processes or tools of the company?

Q12) Does the application work smoothly on different devices and across various browsers?

6.2 Results

This section discusses the results from the think-aloud, questionnaire and the SUS score. Many of the questions were already partially covered when the industrial partner walked through the application and asked what came into his mind; therefore, some redundancy existed afterward when he was posed the questions. The following list gathers all the feedback from the think-aloud part.

1. The CoReTM stencil library should be open by default upon starting the application.
2. Users should only be allowed to modify the label of elements within the library; other characteristics should remain unchangeable.
3. IDs in the threat tables should begin at 1 rather than 0 for clarity and consistency.
4. The description field in the overview table should always display the label of the dataflow instead of being left empty.
5. The button beneath the overview table should display “Save” only if changes have been made after an import; otherwise, it should say “Next”.
6. Adding a button to navigate one step back in the threat modeling process would enhance user control and flexibility.
7. The backgrounds of all labels within the DFD should be set to transparent to avoid confusion when the diagram is rendered as an SVG, where labels might resemble external entities.
8. Text fields within the threat and overview tables should expand when clicked, improving readability beyond the field’s initial size.
9. Increase the whitespace between the threat and overview tables for a cleaner layout.
10. Include a title for the overview table.
11. Add more whitespace between the SVG diagram and the overview table to improve visual separation.

12. Align the title of the text fields in the threat and overview tables to the left for better consistency and readability.
13. The checkboxes within the overview table should have a color contrasting more strongly with the table's blue background.
14. In the threat tables, the icon for adding a new row should be green, and the icon for deleting a row should be red. Alternatively, both icons could be white for uniformity.
15. The borders of the main container should have consistent spacing from the sides, top, and bottom.
16. The menu bar of the Draw.io editor should be hidden to reduce visual clutter.
17. Ensure that all buttons are the same length for a more uniform appearance.
18. Arrange the buttons horizontally rather than vertically for a more streamlined interface.
19. The "Home" button appears unexpectedly after the "Download" button is clicked, which is confusing. Functions should not be hidden; consider disabling it instead or providing feedback that clicking "Home" will result in progress loss.
20. The project name should be displayed as a general title rather than within the iFrame after the DFD is submitted.
21. Upon submitting the overview table, the container loses its top and bottom margins, expanding to fill the entire screen. This should be adjusted for consistency.
22. Additional export formats, such as HTML, PDF, or Markdown, would be highly beneficial for utilizing the threat model after its creation.
23. The application could be further enhanced by including a risk assessment feature, such as DREAD, following the completion of threat modeling.
24. When modeling larger systems, it would be beneficial to be able to collapse the tables to reduce scrolling.
25. The system alerts when the tables are not completely filled out, but it would be helpful to highlight the empty fields then.
26. An "Info" button could be useful for new users who have never interacted with the tool.

Another issue that hasn't been listed is that the import function didn't work on the partner's Linux-based computer when using the Firefox browser. However, this couldn't be replicated on other browsers and machines. It's worth noting that the application was developed using Chrome as the browser, and this information has now been included in the README file for future reference in case someone encounters the same issue.

These points are categorized according to the dimensions outlined in Section 6.1: functionality, completeness, usability, user interface, and integration and compatibility. Each

point is classified within these dimensions and assigned to a specific category. This is depicted in Table 6.1. The possible categories include:

- *UX Improvements*: Enhancements that refine the user experience, making the application more intuitive, user-friendly, and visually appealing. These are often related to the user interface or minor tweaks that improve interaction without altering core functionality.
- *Feature*: New capabilities or functionalities that the application currently lacks. These could be additions that provide more value to the users by expanding the application's capabilities or addressing unmet needs.
- *Bug*: Issues where the application does not function as intended or where existing functionality is broken. Bugs can range from minor visual glitches to significant errors that hinder the application's operation or user experience.

Table 6.1: Categorization of the Feedback

#	Dimension	Category
1	UI, Usability	UX Improvement
2	Functionality, Usability	UX Improvement
3	Usability	UX Improvement
4	Completeness, Usability	UX Improvement
5	Usability	UX Improvement
6	Functionality, UI, Usability	Feature
7	Completeness, UI, Usability	UX Improvement
8	Functionality, UI, Usability	Feature
9	UI, Usability	UX Improvement
10	UI, Usability	UX Improvement
11	UI, Usability	UX Improvement
12	UI, Usability	UX Improvement
13	UI	UX Improvement
14	UI, Usability	UX Improvement
15	UI	UX Improvement
16	Usability	UX Improvement
17	UI	UX Improvement
18	UI	UX Improvement
19	Functionality, Usability	Feature
20	UI	UX Improvement
21	UI	UX Improvement
22	Functionality, Integration & Compatibility	Feature
23	Functionality, Integration & Compatibility, UI, Usability	Feature
24	Functionality, UI, Usability	Feature
25	Functionality, UI, Usability	Feature
26	Functionality, UI, Usability	Feature

Based on the feedback received during the think-aloud process, it was evident that the application met the demands of the industrial partner. There is certainly room for improvement and expansion of the application's capabilities in the future, with 69.2% focusing on UX improvements (18 out of 26), 30.8% on features (6 out of 26), and 0% on bugs (0 out of 26). It's worth noting that no precise UI requirements were specified since the primary goal of this thesis is to evaluate whether the STRIDE-per-Interaction methodology can be partially automated and supported. This may be one reason for the numerous issues in that area, as approximately 73% (19 out of 26) of the problems at least partially target the UI dimension. This analysis aligns quite well with the questionnaire results, which are shown in Table 6.2.

Table 6.2: Answers to the Questionnaire Section 6.1

	Answer
Q1	Only minimally, the IDs of the elements are represented differently but fulfill the same purpose.
Q2	Yes, the tables are being generated. The user is supported as requested.
Q3	This was answered in the previous question and the think-aloud process.
Q4	Refer to the Table 6.1
Q5	Score: 90/100
Q6	The sudden appearance of the "Home" button was confusing. The rest was exactly as specified.
Q7	Yes, it would be ingenious if the export functionality supported different formats.
Q8	Yes, the overview is given. Theoretically, the tables could be collapsed depending on the step the user is in.
Q9	Sometimes, when filling out the tables, feedback is provided. A "Help" button could be useful.
Q10	Incorrect input is hardly possible. Only in the DFD, but there, the user is informed when using illicit elements.
Q11	No, a DREAD analysis and other export formats would be needed for this.
Q12	The file could not be imported using the Firefox browser.

One key aspect covered in the questionnaire but not during the think-aloud process is the SUS (*i.e.*, *Q5*). The SUS evaluation was conducted using a survey, which can be found in Appendix A. SUS measures three key usability aspects: effectiveness (*i.e.*, whether users can achieve their goals using the system), efficiency (*i.e.*, how quickly and easily users can perform tasks), and satisfaction (*i.e.*, how pleasant the experience is) [60]. According to the industrial partner's assessment, *CoReTM 2.0* achieved a SUS score of 90 out of 100, which is exceptionally high. With this score, the application would be graded as A+ and fall in the "Best Imaginable" category [61]. This suggests that the industrial partner is highly satisfied with the system and perceives significant value in it. While the SUS primarily evaluates system usability, the presence of several potential UX improvements and additional features indicates that there is still substantial room for growth. The author believes that, rather than seeing only 10 points left to achieve a "perfect" score, there are broader opportunities for enhancements, which could push the SUS score even higher if addressed effectively.

6.3 Findings

The evaluation of *CoReTM 2.0* indicates that the industrial partner is delighted with the product developed in this project. While there remain opportunities for improvement, the application successfully fulfills its intended purpose and meets the expectations outlined in the provided requirements. Specifically, the software fully supports the application of the STRIDE-per-Interaction methodology and semi-automates several time-consuming tasks. This represents a significant advancement, as discussed in Chapter 3, where it was noted that no such tool previously existed, to the author's knowledge. Although the application is not yet ready for deployment in real-world scenarios – primarily due to the limited export functionality, which currently only supports JSON – it establishes a strong foundation for future enhancements. The repository has been published under the Apache-2.0 license [62], enabling future developers to contribute to the code, create their own implementations, and even use the tool commercially if desired. This contribution marks an essential step toward providing a tool that fully supports the STRIDE-per-Interaction methodology while offering flexibility for community-driven improvements. This thesis could also inspire the development of tools that semi-automate various methodologies using a similar approach by demonstrating that threat modeling methodologies such as STRIDE-per-Interaction can be partially automated. This would benefit the threat modeling domain by significantly reducing the manual effort required, thereby enhancing the efficiency and scalability of the threat modeling process. This could lead to broader adoption of threat modeling techniques in industries where security is a critical concern but resources for such activities are limited.

Additionally, the success of this project may encourage further research and development into automated or semi-automated solutions for other complex security methodologies. By lowering the barrier for implementation, these tools could make sophisticated threat analysis more accessible to a broader range of organizations, fostering a more proactive approach to cybersecurity. Ultimately, this would improve security practices across various sectors, as businesses of all sizes could benefit from streamlined threat identification and mitigation processes. In conclusion, while there are still areas for enhancement, the developed tool already represents a crucial step forward in the field of semi-automated threat modeling. It addresses a significant gap, as noted in Chapter 3, and opens new avenues for future innovations that could revolutionize how threat modeling is conducted in practice.

Chapter 7

Summary

In summary, this thesis contributes to the domain of threat modeling by delivering a functional implementation of the CoReTM framework and offering a semi-automated application of the STRIDE-per-Interaction methodology. This methodology, which focuses on identifying security threats arising from the interactions between different system components, requires the system to be modeled in a DFD.

This is enabled by embedding Draw.io within *CoReTM 2.0*, where users can construct diagrams using a custom stencil library specifically designed for this purpose. Once the DFD is created, the software steps in to automate parts of the process. It parses the diagram and generates an overview table – traditionally a labor-intensive task – by extracting and organizing key information from the diagram elements, such as metadata related to coordinates, labels, and connections established through data flows. This internal representation of the DFD allows the software to deduce the dimensions and structure of the overview table, thus streamlining the process and minimizing manual input. The semi-automation extends further by supporting the generation of threat tables after the overview table has been completed, once again reducing user effort by automating repetitive tasks.

The evaluation of *CoReTM 2.0* was conducted using a user-centered approach in collaboration with an industrial partner. The assessment yielded positive results, with no bugs identified and a high SUS score indicating a high level of satisfaction. The current version of the software has made progress, but it is not yet fully ready for deployment in real-world scenarios. It requires further development to incorporate several important features that are discussed in the future work section 7.1.

Notably, the industrial partner confirmed that the key requirements were met. However, some new features emerged during development, such as rendering the DFD as an SVG upon submission and organizing threat tables based on trust boundaries rather than rows from the overview table. These additional features, while valuable, were not part of the original requirements and emerged during the development process, leading the author to adapt the implementation accordingly. Overall, the thesis lays a strong foundation for further development, with the semi-automation of threat modeling being a promising step towards improving efficiency in identifying and addressing security risks.

7.1 Future Work

As outlined in Section 6.1, there are several key areas for future improvements to *CoReTM 2.0* that were identified during the evaluation phase and are summarized in Table 6.1. While not all issues in the table are equally important, all of them would enhance or expand the application's capabilities. For instance, certain UX improvements, such as left-aligning some columns in the overview and threat tables, adjusting colors for better contrast, and ensuring that IDs in the threat tables start at 1 rather than 0, may be considered non-critical when compared to others. In this section, the focus is on two enhancements that would significantly improve the usability of the application in a real-world scenario.

The most critical enhancement suggested by the industrial partner is the ability to export threat models in more user-friendly formats, such as HTML, PDF, or Markdown. Implementing this functionality would significantly increase *CoReTM 2.0*'s practical utility, allowing the partner to immediately use the tool for generating client reports. Another potential improvement is related to text field usability, particularly when importing existing models. Currently, text fields do not display their full content, making it difficult to review imported information. A potential solution could involve expanding text fields or enabling a pop-up text box when clicking on a table cell, allowing for easier reading and interaction.

In addition to these primary features, there are numerous opportunities to improve the UX across the application. Beyond functional improvements, the project is designed with future adaptability in mind. The code has been published under the Apache-2.0 license, which permits future developers to build upon it without concerns about copyright infringement. This licensing choice aims to foster ongoing development, encouraging others to refine or extend the application's core features and functionalities to suit their specific needs, ensuring the long-term evolution and relevance of *CoReTM 2.0*.

Bibliography

[1] A. Bendovschi, “Cyber-Attacks – Trends, Patterns and Security Countermeasures”, *Procedia Economics and Finance*, vol. 28, pp. 24–31, Apr. 2015.

[2] Bueermann, Gretchen and Rohrs, Michael, *Global Cybersecurity Outlook 2024*, Last Accessed: 09.09.2024, Jan. 2024. [Online]. Available: https://www3.weforum.org/docs/WEF_Global_Cybersecurity_Outlook_2024.pdf.

[3] Z. Shi, K. Graffi, D. Starobinski, and N. Matyunin, “Threat Modeling Tools: A Taxonomy”, *IEEE Security & Privacy*, vol. 20, no. 4, pp. 29–39, Dec. 2022.

[4] A. Shostack, *Threat Modeling: Designing for Security*, 1st. John Wiley & Sons, Inc., Feb. 2014.

[5] J. Von Der Assen, M. F. Franco, C. Killer, E. J. Scheid, and B. Stiller, “CoReTM: An Approach Enabling Cross-Functional Collaborative Threat Modeling”, in *2022 IEEE International Conference on Cyber Security and Resilience (CSR)*, IEEE, Rhodes, Greece, Jul. 2022, pp. 189–196.

[6] Jan von der Assen, *Collaborative and Remote Threat Modeling*, Last Accessed: 27.05.2024, Jan. 2022. [Online]. Available: https://github.com/jvdassen/coretm.

[7] M. Howard and S. Lipner, *The security development lifecycle*. Microsoft Press, Jun. 2006, vol. 8.

[8] D. B. Parker, *Fighting computer crime: A new framework for protecting information.* John Wiley & Sons, Inc., Aug. 1998.

[9] L. O. Nweke and S. D., “A Review of Asset-Centric Threat Modelling Approaches”, *International Journal of Advanced Computer Science and Applications*, vol. 11, no. 2, 2020.

[10] M. Corporation, *Improving web application security: threats and countermeasures.* Microsoft Press, Sep. 2003.

[11] A. Amini, N. Jamil, A. R. Ahmad, and M. R. Zaba, “Threat Modeling Approaches for Securing Cloud Computing”, *Journal of Applied Sciences*, vol. 15, no. 7, pp. 953–967, Jul. 2015.

[12] T. UcedaVelez and M. M. Morana, *Risk Centric Threat Modeling: process for attack simulation and threat analysis.* John Wiley & Sons, May 2015.

[13] M. Abomhara, M. Gerdes, and G. M. Køien, “A STRIDE-Based Threat Model for Telehealth Systems”, *Norsk informasjonssikkerhetskonferanse (NISK)*, vol. 8, no. 1, pp. 82–96, Nov. 2015.

[14] L. Kohnfelder and P. Garg, "The threats to our products", *Microsoft Interface, Microsoft Corporation*, vol. 33, Apr. 1999.

[15] F. Swiderski and W. Snyder, *Threat modeling*. Microsoft Press, Jul. 2004.

[16] P. Torr, "Demystifying the threat modeling process", *IEEE Security & Privacy*, vol. 3, no. 5, pp. 66–70, Oct. 2005.

[17] Microsoft Inc., *Uncover Security Design Flaws Using The STRIDE Approach*, Last Accessed: 27.04.2024, Jul. 2019. [Online]. Available: https://learn.microsoft.com/en-us/archive/msdn-magazine/2006/november/uncover-security-design-flaws-using-the-stride-approach.

[18] CAIRIS, *CAIRIS: threat modelling with DFDs and attack trees*, Last Accessed: 09.05.2024, Aug. 2019. [Online]. Available: https://www.youtube.com/watch?v=kJ2NUelcM_o.

[19] S. Faily, *Designing usable and secure software with IRIS and CAIRIS*. Springer, Apr. 2018.

[20] K. Tan and V. Garg, *An analysis of open-source automated threat modeling tools and their extensibility from security into privacy*, Last Accessed: 12.09.2024, Feb. 2022. [Online]. Available: https://www.usenix.org/publications/loginonline/analysis-open-source-automated-threat-modeling-tools-and-their.

[21] S. Faily and C. Iacob, "Design as Code: Facilitating Collaboration Between Usability and Security Engineers Using CAIRIS", in *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, IEEE, Lisbon, Portugal, Sep. 2017, pp. 76–82.

[22] CAIRIS, *Computer Aided Integration of Requirements and Information Security - Server*, Last Accessed: 30.05.2024, May 2024. [Online]. Available: https://github.com/cairis-platform/cairis.

[23] CAIRIS, *Starting CAIRIS*, Last Accessed: 30.05.2024, Dec. 2021. [Online]. Available: https://cairis.readthedocs.io/en/latest/starting.html.

[24] Jan von der Assen, *MetaProcess dizmo*, Last Accessed: 27.05.2024, Jun. 2022. [Online]. Available: https://github.com/jvdassen/coretm-process.

[25] J. von der Assen, M. F. Franco, C. Killer, E. J. Scheid, and B. Stiller, "On collaborative threat modeling", *IFI-TecReport No. 2022.04, Zürich, Switzerland, Tech. Rep.*, Apr. 2022.

[26] Jonny Tennyson, *STRIDE and CAPEC with IriusRisk*, Last Accessed: 06.05.2024, Jun. 2022. [Online]. Available: https://www.iriusrisk.com/resources-blog/stride-and-capec-with-iriusrisk.

[27] IriusRisk, *The Open Threat Modeling platform*, Last Accessed: 27.05.2024, Sep. 2023. [Online]. Available: https://github.com/iriusrisk/Community.

[28] IriusRisk, *Pricing*, Last Accessed: 27.05.2024. [Online]. Available: https://www.iriusrisk.com/plans.

[29] IriusRisk, *Threat Modeling Methodologies*, Last Accessed: 27.05.2024. [Online]. Available: https://www.iriusrisk.com/threat-modeling-methodologies.

[30] David Doughty, *Applying STRIDE Methodology to Threat Model a New Component*, Last Accessed: 27.05.2024, Apr. 2023. [Online]. Available: `https://www.iriusrisk.com/resources-blog/applying-stride-methodology-to-threat-model-a-new-component`.

[31] Rhys McNeill, *IriusRisk Installation Guide*, Last Accessed: 03.06.2024, Jun. 2024. [Online]. Available: `https://support.iriusrisk.com/hc/en-us/articles/16491409456541-IriusRisk-Installation-Guide`.

[32] E. A. AbuEmera, H. A. ElZouka, and A. A. Saad, "Security Framework for Identifying threats in Smart Manufacturing Systems Using STRIDE Approach", in *2022 2nd International Conference on Consumer Electronics and Computer Engineering (ICCECE)*, Guangzhou, China: IEEE, Jan. 2022, pp. 605–612.

[33] Y. U. Mulla, A. Chavekar, S. Mane, and F. Kazi, "Threat Modeling of Cube Orange Based Unmanned Aerial Vehicle System", in *2023 IEEE International Carnahan Conference on Security Technology (ICCST)*, Pune, India: IEEE, Oct. 2023, pp. 1–6.

[34] L. H. Flå, R. Borgaonkar, I. A. Tøndel, and M. Gilje Jaatun, "Tool-assisted Threat Modeling for Smart Grid Cyber Security", in *2021 International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA)*, Dublin, Ireland: IEEE, Jun. 2021, pp. 1–8.

[35] A. Karahasanovic, P. Kleberger, and M. Almgren, "Adapting threat modeling methods for the automotive industry", in *Proceedings of the 15th ESCAR Conference*, Berlin, Germany, Nov. 2017, pp. 1–10.

[36] E. Bygdås, L. A. Jaatun, S. B. Antonsen, A. Ringen, and E. Eiring, "Evaluating Threat Modeling Tools: Microsoft TMT versus OWASP Threat Dragon", in *2021 International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA)*, Dublin, Ireland: IEEE, Jun. 2021, pp. 1–7.

[37] Microsoft Inc., *Microsoft Threat Modeling Tool*, Last Accessed: 28.05.2024, Aug. 2022. [Online]. Available: `https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool`.

[38] K. Tuma and R. Scandariato, "Two architectural threat analysis techniques compared", in *Software Architecture: 12th European Conference on Software Architecture, Proceedings 12*, Springer, Madrid, Spain, Sep. 2018, pp. 347–363.

[39] Microsoft Inc., *Threat Modeling Tool Releases*, Last Accessed: 28.05.2024, Jan. 2023. [Online]. Available: `https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool-releases`.

[40] OWASP Foundation Inc., *OWASP Threat Dragon*, Last Accessed: 28.05.2024, Feb. 2024. [Online]. Available: `https://owasp.org/www-project-threat-dragon/docs-2/about/`.

[41] Open Security Summit, *Lightning Demo - Threatmodel Tool Demos by Steven Wierckx and Mike Goodwin - 16 Jun*, Last Accessed: 28.05.2024, Jun. 2020. [Online]. Available: `https://www.youtube.com/watch?v=n6JGcZGFq5o`.

[42] Christian Schneider, *Agile Threat Modeling Toolkit*, Last Accessed: 29.05.2024, May 2024. [Online]. Available: `https://github.com/Threagile/threagile`.

[43] Christian Schneider, *Threagile - Agile Threat Modeling*, Last Accessed: 29.05.2024, 2020. [Online]. Available: `https://threagile.io/`.

[44] M. Sedlmair, M. Meyer, and T. Munzner, "Design study methodology: Reflections from the trenches and the stacks", *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 12, pp. 2431–2440, Dec. 2012.

[45] JGraph Ltd and draw.io AG, *Draw.io*, Last Accessed: 04.06.2024, Jun. 2024. [Online]. Available: `https://github.com/jgraph/drawio`.

[46] JGraph Ltd and draw.io AG, *About draw.io*, Last Accessed: 04.06.2024. [Online]. Available: `https://www.drawio.com/about`.

[47] JGraph Ltd and draw.io AG, *Features of draw.io*, Last Accessed: 04.06.2024. [Online]. Available: `https://www.drawio.com/features`.

[48] Microsoft Inc., *The TypeScript Handbook*, Last Accessed: 19.08.2024. [Online]. Available: `https://www.typescriptlang.org/docs/handbook/intro.html`.

[49] Stack Exchange, *Programming, scripting, and markup languages*, Last Accessed: 22.07.2024, Jul. 2024. [Online]. Available: `https://survey.stackoverflow.co/2024/technology#2-programming-scripting-and-markup-languages`.

[50] Meta Platforms Inc., *The library for web and native user interfaces*, Last Accessed: 20.08.2024. [Online]. Available: `https://react.dev/`.

[51] JGraph Ltd and draw.io AG, *Work with custom shape libraries*, Last Accessed: 22.07.2024, Apr. 2020. [Online]. Available: `https://www.drawio.com/blog/custom-libraries`.

[52] Jan, von der Assen, *ThreatFinder*, Last Accessed: 22.07.2024. [Online]. Available: `https://threatfinder-ai.pages.dev/`.

[53] Jan von der Assen, *Threat Modeling for AI Systems*, Last Accessed: 22.07.2024, Jun. 2024. [Online]. Available: `https://github.com/jvdassen/ThreatFinder.ai`.

[54] MUI, *Material UI: React components that implement Material Design*, Last Accessed: 22.07.2024. [Online]. Available: `https://mui.com/material-ui/`.

[55] Mozilla Foundation, *Web Storage API*, Last Accessed: 20.08.2024. [Online]. Available: `https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage`.

[56] Remix Software Inc., *React Router*, Last Accessed: 23.07.2024. [Online]. Available: `https://reactrouter.com/en/main`.

[57] JSON.org, *Introducing JSON*, Last Accessed: 20.08.2024. [Online]. Available: `https://www.json.org/json-en.html`.

[58] JGraph Ltd and draw.io AG, *Embed mode*, Last Accessed: 29.07.2024. [Online]. Available: `https://www.drawio.com/doc/faq/embed-mode`.

[59] J. Karat, "User-Centered Software Evaluation Methodologies", in *Handbook of Human-Computer Interaction*, M. G. Helander, T. K. Landauer, and P. V. Prabhu, Eds., Second Edition. North-Holland, 1997, pp. 689–704.

[60] J. Brooke, "SUS: A quick and dirty usability scale", *Usability Evaluation in Industry*, vol. 189, Nov. 1995.

[61] Jeff Sauro, *5 Ways to Interpret a SUS Score*, Last Accessed: 09.09.2024, Sep. 2018. [Online]. Available: `https://measuringu.com/interpret-sus-score/`.

[62] Apache Software Foundation, *Apache License, Version 2.0*, Last Accessed: 05.09.2024, Jan. 2004. [Online]. Available: `https://www.apache.org/licenses/LICENSE-2.0`.

Abbreviations

CAIRIS	Computer Aided Integration of Requirements and Information Security
CSS	Cascading Style Sheets
DFD	Data Flow Diagram
DoS	Denial of Service
JSON	JavaScript Object Notation
MTMT	Microsoft Threat Modeling Tool
MUI	Material UI
MVP	Minimum Viable Product
SUS	System Usability Scale
TD	Threat Dragon
UX	User Experience
UI	User Interface

List of Figures

2.1 Security Development Lifecycle Process Diagram 4
2.2 DFD Symbols extension to [4] 6
2.3 DFD of an Example System 7

4.1 Architectural Diagram of the *CoReTM 2.0* MVP 17

5.1 Visualization of Element's Calculation Within Trust Boundary 31
5.2 Landing Page *CoReTM 2.0* MVP 34
5.3 Define Project Name 35
5.4 Drawio Embedding 35
5.5 Overview Table 36
5.6 Threat Tables 36
5.7 Download Threat Model 37
5.8 Rendering SVG after DFD completion 38

List of Tables

2.1 STRIDE-per-Interaction: Threat Applicability Overview 7

3.1 Comparison of Different Threat Modeling Tools 10

4.1 Overview Requirements for the MVP Prototype 16

6.1 Categorization of the Feedback 47

6.2 Answers to the Questionnaire Section 6.1 48

Listings

5.1 Showcase different Links to Routes 21
5.2 Definition of the Routes within the Application 22
5.3 Definition of the IElement Interface 23
5.4 Definition of the ITrustBoundary Interface 23
5.5 Definition of the IDataFlow Interface 23
5.6 Definition of the IDiagram Interface 24
5.7 Definition of the ICrossingElements Interface 24
5.8 Definition of the IOverviewTableRow Interface 24
5.9 Definition of the IThreatTableRow Interface 25
5.10 Exporting Local State to JSON and Invoking Download 26
5.11 DrawioController Constructor 26
5.12 IFrame of Draw.io embedding 27
5.13 Incoming Events from Draw.io 28
5.14 Parsing invocation within the DrawioController 29
5.15 Parsing the XMLDocument within the DiagramAnalyser Class 30
5.16 2D Calculation to determine if Element is in Trust Boundary 31
5.17 Function Definition of the OverviewTable Component 32
5.18 Function Definition of the ThreatTables Component 33
5.19 ImportController Class Definition 40
5.20 Logic for Handling the Next Button Click 41
5.21 Update generated Threat Tables from Import Data 42

Appendix A

System Usability Scale

app.surveylab.com

COUNTRY

en

LANGUAGE

2024/08/28 09:51:36

DATE

26124380

RESPONSE KEY

1 min. 16 sec.

TIME SPENT

Desktop

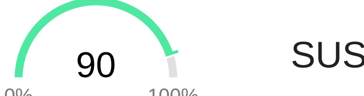
DEVICE

Windows 10.0

SYSTEM

Chrome 113.0.5666.197

BROWSER

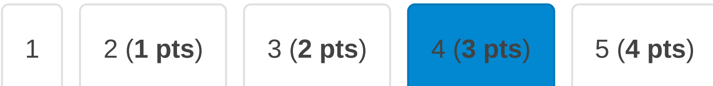


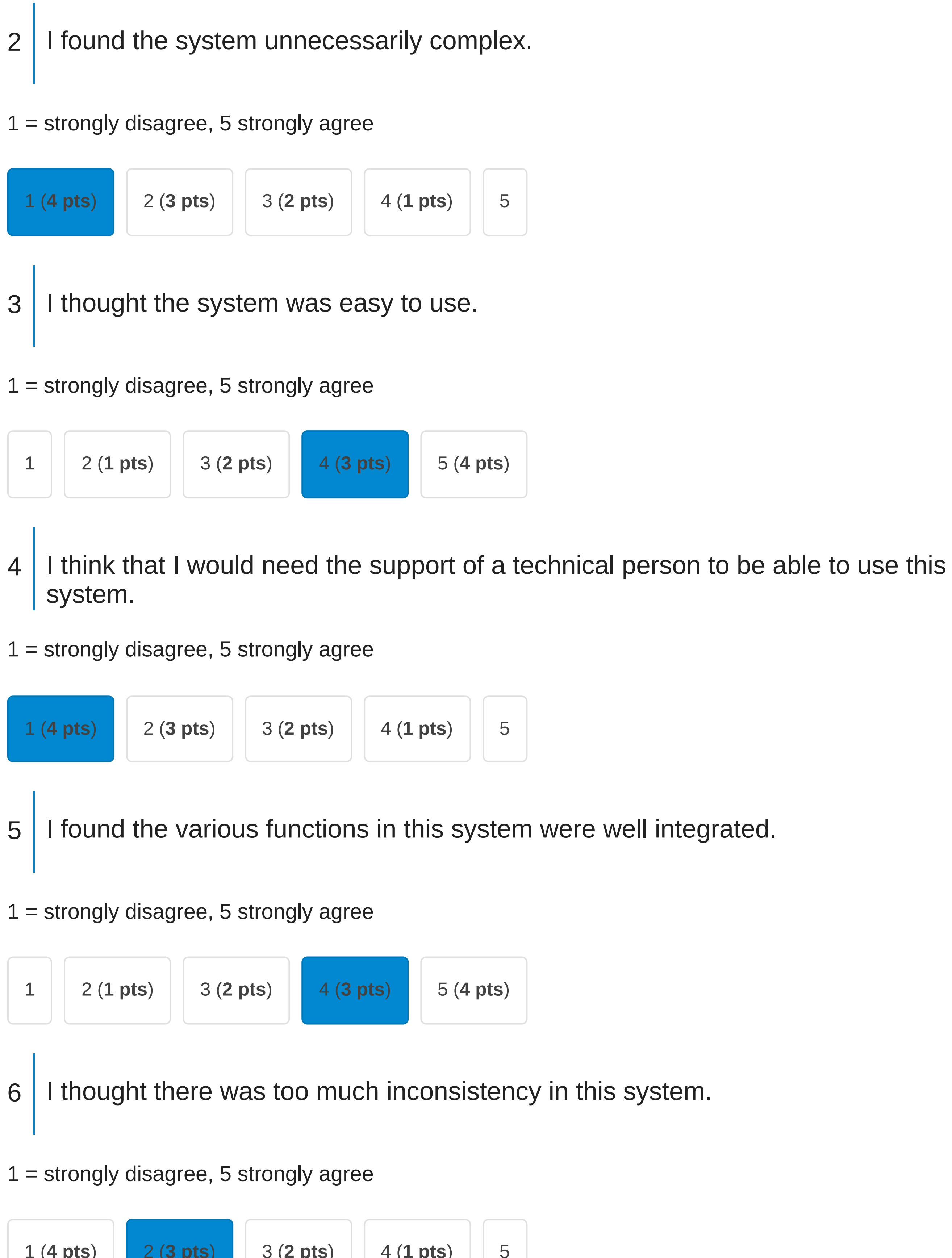
SUS

Nazwa Strony

1 | I think that I would like to use this system frequently.

1 = strongly disagree, 5 strongly agree





2 I found the system unnecessarily complex.

1 = strongly disagree, 5 strongly agree

1 (**4 pts**) | 2 (**3 pts**) | 3 (**2 pts**) | 4 (**1 pts**) | 5

3 I thought the system was easy to use.

1 = strongly disagree, 5 strongly agree

1 | 2 (**1 pts**) | 3 (**2 pts**) | 4 (**3 pts**) | 5 (**4 pts**)

4 I think that I would need the support of a technical person to be able to use this system.

1 = strongly disagree, 5 strongly agree

1 (**4 pts**) | 2 (**3 pts**) | 3 (**2 pts**) | 4 (**1 pts**) | 5

5 I found the various functions in this system were well integrated.

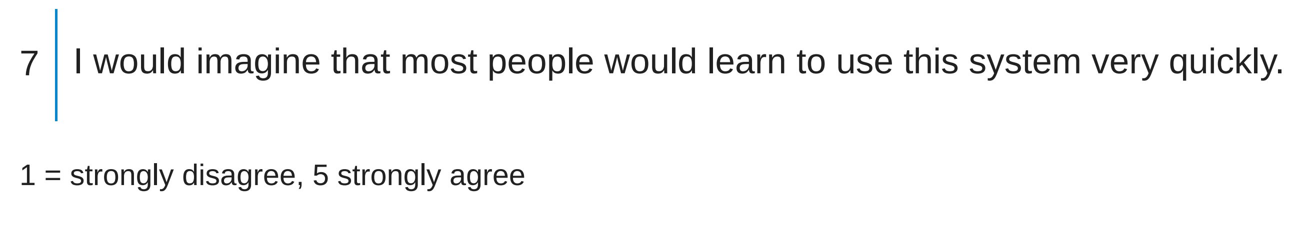
1 = strongly disagree, 5 strongly agree

1 | 2 (**1 pts**) | 3 (**2 pts**) | 4 (**3 pts**) | 5 (**4 pts**)

6 I thought there was too much inconsistency in this system.

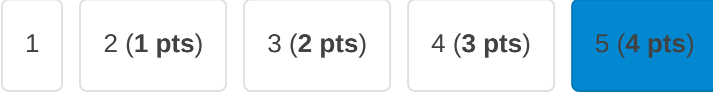
1 = strongly disagree, 5 strongly agree

1 (**4 pts**) | 2 (**3 pts**) | 3 (**2 pts**) | 4 (**1 pts**) | 5



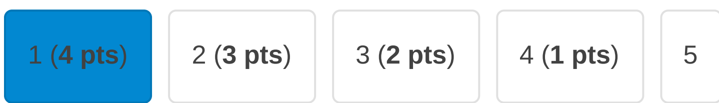
7 I would imagine that most people would learn to use this system very quickly.

1 = strongly disagree, 5 strongly agree



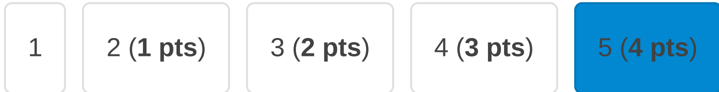
8 I found the system very cumbersome to use.

1 = strongly disagree, 5 strongly agree



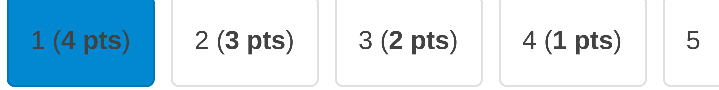
9 I felt very confident using the system.

1 = strongly disagree, 5 strongly agree



10 I needed to learn a lot of things before I could get going with this system.

1 = strongly disagree, 5 strongly agree



Appendix B

Installation Guidelines

The installation guidelines as well as the source code can be found at:

https://github.com/mirovv/CoReTM-2.0

Appendix C

Contents of the CD

- Thesis (.pdf)
- Midterm Presentation (.pptx)
- Final Presentation (.pptx)
- Overleaf Project (.zip)
- Source Code (.zip)