Politecnico di Torino

Master of Science in Cybersecurity

Master Degree Thesis

Design and implementation of a threat modeling tool for data-sharing ecosystems

Supervisors
prof. Fulvio Valenza
prof. Daniele Bringhenti
prof. Riccardo Sisto
dott. Francesca Coriale

Candidate
Gaetano PENNAVARIA

Academic Year 2025-2026

This work is subject to the Creative Commons Licence

Summary

This thesis addresses the need to detect security threats in the data-sharing ecosystem. In the modern digital era, digital services rely on continuous data exchange between users, platforms or any kind of service related. This evolution allows new services such as personalized ads, collaboration and platform interoperability, but this poses new risks for the user. Starting from a simple photo shared on a social network may contribute to high-impact inferences, and actors with different trust levels may interact with data whose privacy impact is not always explicit. It is straightforward to realize that manual analysis of data flows and sharing is difficult to scale, it may fail to capture indirect access paths, third-party propagation, and aggregation effects.

In order to address this manual problem, the main goal of this thesis is to propose a formal threat model for data-sharing ecosystems. The model describes the main elements of the data-sharing ecosystem and the possible elements that can be exploitable by an attack, starting from these, the model derives whether a data item becomes accessible and whether a harmful condition can occur.

The model is implemented in Prolog, using a modular structure that separates the general derivation rules from the facts of each specific scenario. This allows users to insert new scenarios and analyze them. In order to validate the model, different cases are implemented which range across different scenarios. Finally, a GUI (graphical user interface) is implemented to visually show the possible attack paths of the attacker.

The results confirm that, by sharing different data through multiple services, their aggregation can lead to the derivation of new knowledge, and may enable new attack paths that would not be visible when analyzing services individually.

Acknowledgements

Contents

List of Figures 8

List of Tables 9

Listings 10

1 Introduction 11

2 Data Sharing Ecosystem 13

2.1 Introduction to Data Sharing . . . 13

2.2 Legal and Regulatory Context . . . 15

2.2.1 GDPR as the Main Legal Framework . . . 15

2.2.2 Main Actors in Personal Data Processing . . . 15

2.2.3 Personal Data and Identifiability . . . 16

2.2.4 Data Sharing as Controlled Processing . . . 17

2.3 Architecture of Consumer Data Sharing Platforms . . . 18

3 Threat Modeling 21

3.1 Definitions . . . 21

3.2 Attack trees and Attack graphs . . . 22

3.3 Threat Modeling Methodologies . . . 23

3.3.1 STRIDE . . . 23

3.3.2 MITRE ATT&CK . . . 24

3.3.3 PASTA . . . 25

3.3.4 LINDDUN . . . 26

3.3.5 TAMELESS . . . 27

3.3.6 Comparative analysis . . . 29

3.4 Advantages and Challenges . . . 29

3.5 Process and Scope . . . 31

3.5.1 Example of Threat Model Process . . . 32

3.6 From Threat Modeling to Automated Countermeasures . . . 38

4 Thesis Objectives 40
4.1 Thesis Workflow . . . 40

5 Approach and Model 42
5.1 Overview of the Proposed Approach . . . 42
5.2 Entities . . . 42
5.3 Relation between entities . . . 43
5.4 Properties . . . 43
5.4.1 Assumed Property . . . 44
5.4.2 Derived Properties . . . 44
5.5 Derivation rules . . . 44
5.5.1 Trust Policy Rules . . . 45
5.5.2 Rules for accessibility . . . 45
5.5.3 Harm Rule . . . 46
5.6 Application of the model in a use case . . . 46

6 Implementation 49
6.1 Architecture Overview . . . 49
6.2 Prolog Engine . . . 50
6.2.1 Prolog . . . 50
6.3 Backend . . . 52
6.3.1 Python . . . 52
6.3.2 Running the model . . . 52
6.3.3 Queries . . . 54
6.3.4 Generating cases and recovering facts . . . 54
6.3.5 Building the attack graph . . . 56
6.3.6 REST API . . . 57
6.4 Frontend . . . 58
6.4.1 React and Vite . . . 59
6.4.2 Rest client . . . 59
6.4.3 Attack graph rendering . . . 61
6.4.4 Presenting the results and building cases . . . 63

7 Validation 66
7.1 First use-case . . . 66
7.1.1 Scenario description . . . 66
7.1.2 Prolog implementation . . . 66
7.1.3 Results obtained . . . 68
7.2 Second use-case . . . 71
7.2.1 Scenario description . . . 71
7.2.2 Prolog implementation . . . 71
7.2.3 Results obtained . . . 72

8 Conclusions 76
8.1 Limitations and Future Works . . . 76

Bibliography 78

List of Figures

2.1 Simplified data-flow architecture of a consumer data-sharing platform 19

3.1 STRIDE Threat Model [1] 24
3.2 Data Flow Diagram of the scenario 33

6.1 High-level architecture of the tool 50
6.2 Home page with use cases to be selected 61
6.3 Statistics, graph and harm table example 64
6.4 Form to create an use case 65

7.1 Statistics of the first case 68
7.2 Assumed and derived properties of the first case 69
7.3 Query identifying who can harm a data item 69
7.4 Attack graph output 70
7.5 Attack path of Cambridge Analytica to psychographic profile 70
7.6 Statistics of the second case 72
7.7 Assumed and derived properties of the second case 73
7.8 Query on which data items an actor can harm 73
7.9 Attack graph output 74
7.10 Daily movement path 75
7.11 Home location path 75

List of Tables

2.1 Main PII categories relevant to consumer platforms, derived from the examples provided in NIST SP 800-122 [2] 16
2.2 GDPR provisions relevant to data sharing [3] 17

3.1 Seven stages of the PASTA methodology 26
3.2 LINDDUN privacy threat types 27
3.3 Comparison of the threat modeling methodologies 29
3.4 Mapping of the threat modeling question to the OWASP process step 32
3.5 Elements represented in the scenario DFD 34
3.6 Data flows represented in the scenario DFD 34
3.7 Main assets in the scenario 35
3.8 Mapping of the threats using LINDDUN 36
3.9 Possible mitigations for the identified privacy threats 37

5.1 Derivation rules for trust 45
5.2 Derivation rules for accessibility 45
5.3 Derivation rule for harm 46
5.4 Description of the facts used in the worked example 47

6.1 Main REST endpoints 58

Listings

6.1 Rules implemented in base_model.pl 51
6.2 Running the prolog engine as a subprocess (`prolog_engine.py`) . . 52
6.3 EMIT in prolog_engine.py 53
6.4 Parse function in prolog_engine.py 53
6.5 Function query_case_text in prolog_engine.py 54
6.6 Function form_to_pl in prolog_engine.py 54
6.7 Function parse_facts in prolog_engine.py 55
6.8 Function build_graph in graph.py 56
6.9 Analyze function and a REST endpoint in app.py 58
6.10 vite.config.js 59
6.11 api.js 59
6.12 The application shell and the open-case handler in App.jsx 59
6.13 Function Graph in Graph.jsx 61
7.1 Implementation of Cambridge analytica case 67
7.2 Implementation of the second use-case 71

Chapter 1

Introduction

Today's digital era is characterized by the constant flow of personal data, a simple action like uploading an image or providing a link to a document normally impacts more than one person. Instead, it triggers a chain reaction in which it is hard to predict who will be affected at last. When a user decides to share information, they start to reason about one data item a time, but it assumes different meaning based on the context of sharing. Data in one context may become accessible in another, or data that appear harmless in isolation may reveal more sensitive information when combined with other data. So it is not enough to consider data sharing as a simple technical exchange, instead it must be analyzed as a process involving platforms, actors, access conditions, trust assumptions and possible derived information. Usually in consumer platform, users may see only a button to publish content, but behind this action platforms can store the data, propagate them to other services or combine them with additional information. This leads to understanding that privacy and security threats may arise not only from explicit attacks, but also from normal data-sharing actions and on how the data flows are organized.

The problem is understanding how these data flows work and it is impractical to do it manually. There are a huge amount of possible combinations and it grows rapidly with the number of data items and sharing relationships. What is missing is a systematic and automatic way to determine, from a description of how a system shares data, which information ultimately becomes exposed and which actors are in a position to cause harm.

This thesis addresses that problem by proposing a logic-based approach to a formal threat model for data sharing. The main entities of the model are actors, data collectors, data items, privacy-impact levels, trust levels, and attacks. These entities are combined with relation which describe how impact and trust are assigned and how data is shared, propagated, forwarded, and aggregated. Finally, a set of derivation rule is made that captures the behavior of the data-sharing ecosystem. The proposed model was then implemented in Prolog, whose resolution mechanism is naturally suited to this kind of rule-based inference, and is made accessible through a web application that lets the user describe a system and then visualizes the results as an attack graph, together with the individual attack paths that lead to each harm, and finally validated with a set of use cases. First of all

there is the description of a system: the data items and their privacy impact, the actors and their trust levels, and then the ways in which data items are shared and the attacks that make exploitable a data item. The tool automatically derives which data items become accessible and which actors can actually cause harm. As mentioned before, the tool has been validated on a set use cases which refer to different data-sharing ecosystem, this allows to see how the tool behaves in different scenarios.

The goal of this thesis is to provide a structured and formal way to analyze how threats emerge in data-sharing ecosystems, in particular it aims to support the reasoning phase that identifies how data shared through different services can derive new knowledge in order to enable new attack path which were not visible at the beginning.

Chapter 2

Data Sharing Ecosystem

2.1 Introduction to Data Sharing

One definition of data sharing can be the process through which data are made available among different actors. These actors may be individuals, platforms, or any kind of service by which it is possible to exchange information. So in a digital environment it is possible to say that data sharing is related to the continuous circulation of data through online services. Today, in the digital era, the relevance of data sharing has grown, a lot of systems rely on the possibility of collecting and exchanging information in order to provide different services based on the information acquired.

The European strategy for data describes data as an important factor which can contribute to economic growth, job creation or societal progress, while also stressing that digital development should remain focused on people and European values [4]. It is possible to affirm that data sharing is not only a technical operation but also a key component of the contemporary digital economy.

However, data sharing creates important privacy risks and challenges because once data are shared they may be accessed, copied, stored, reused, transferred, or combined with other information and this creates some problems for individuals to understand how their data are managed. The OECD Privacy guidelines already identify principles such as purpose specification, use limitation, security safeguards, openness, individual participation, and accountability as essential safeguards for the circulation of personal data [5].

In some consumer platforms, data sharing is fundamental. In social networks, it is a normal activity for the user to upload photos and send messages, but also other related activities. The point is that there exist different forms of data sharing that are not visible since they are part of the technical functioning of the platform, for example, through analytics or advertising systems.

It is possible to make a distinction between two levels of data sharing:

- **User-facing sharing**, where the user intentionally performs an action such as publishing, sending or related actions.

- **Infrastructure-level sharing**, where data circulate through backend services, cloud storage or related services.

Looking at this distinction, it is possible to understand that the user can see only the first level while the platform interface may show, for example, only a simple button. Thanks to that action, data can be stored, indexed, synchronized, logged, analyzed, or made available to other technical components.

The NIST Privacy Framework adopts a broad view of privacy risk management and refers to the need to understand data processing activities within a wider data processing ecosystem [6]. This perspective is useful because it avoids reducing privacy to a single moment of disclosure.

As explained above data sharing can take different forms depending on the context. In social networks where a user creates posts, comments, interactions. In cloud services, it may involve documents, folders, backups and collaborative editing. In location-based services, it may involve routes, reviews, check-ins, or mobility patterns. In productivity platforms, it may involve calendars, meetings, contacts, files, and shared workspaces. In each case, data sharing allows digital services to operate, but also increases the amount of information that may circulate beyond the user's immediate control.

There are a lot of advantages of data sharing, it supports communication, collaboration, research, service improvement, interoperability, and innovation. Different examples can be made, it is possible to improve digital service by using shared data, allow users to collaborate on documents and also it can contribute to the development of the AI systems. There are a lot of benefits that come from it and it should not be understood only as a source of risk [7].

However the same mechanism cited before creates exposure. A data item shared in one context may be reused in another. Some information disclosed to a limited audience may become accessible through forwarding, links, platform settings, or third-party services. Some normal data may have limited significance when considered in isolation, however when data associated with identifiers, metadata, location, or time they can show information which was not apparent from the original data. According to [8], identification is not limited to retrieving a name or address, but also includes identifiability through singling out, linkability and inference, for example in consumer platforms there are many services that are connected through the same account, which makes the linkage easier.

Some challenges arise in data sharing but the main one is that this kind of ecosystem changes based on the conditions under which information can be accessed and interpreted. Once a data item enters a platform ecosystem, it may be processed in ways that are not immediately visible to the user. This means that different actions could be performed on it without the user knowing, and this creates a problem about the user's perception of a single sharing action and the broader lifecycle of the data.

For this reason, the study of data sharing requires attention to three general dimensions:

- **Technical dimension**: how data are transmitted, stored, synchronized, accessed, and processed by platforms and services.

- **Organizational dimension**: which actors manage, receive, reuse, or govern the data.
- **Privacy dimension**: how sharing affects the individual's control, expectations, exposure, and possible identifiability.

These dimensions allow to understand that a data sharing ecosystem can be seen as a big socio-technical environment in which personal data are generated, collected, exchanged, processed, reused, and correlated by multiple actors. A complete analysis of data sharing must therefore consider not only the act of making data available, but also the environment in which the data circulate and the responsibilities of the actors involved.

2.2 Legal and Regulatory Context

2.2.1 GDPR as the Main Legal Framework

When personal data are shared through different platforms, from a legal point of view in the European Union the reference is Regulation (EU) 2016/679, the General Data Protection Regulation (GDPR), which regulates the processing of personal data and defines the main roles, principles, and safeguards that govern data flows [3].

There are three main reasons why it is relevant in this work:

- it defines what counts as personal data;
- it identifies the principal actors involved in data processing;
- although it does not use the term "data sharing" explicitly, it frames the operations it involves as processing, which means that technical operations performed on data such as access, storage or combination of data are legally relevant actions.

2.2.2 Main Actors in Personal Data Processing

The GDPR uses a data-centric approach in which it is possible to define three main actors:

- **Data subject**: the natural person to whom the personal data relate;
- **Data controller**: the entity that determines the purposes and means of the processing;
- **Data processor**: the entity that processes personal data on behalf of the controller.

Since in data sharing there are multiple actors, the concept of roles is important. A user may generate or disclose data through a platform; the platform may determine how those data are collected, stored, used, and shared; and external providers may process the data on behalf of the platform. The GDPR defines the controller as the entity that decides the purposes and means of processing, while the processor acts on behalf of the controller [3, Art. 4(7), 4(8)]. The EDPB further clarifies that these are functional roles: what matters is the factual influence over the processing, not only the formal label used in a contract [9].

In consumer platforms, the platform itself can act as controller when it decides, for example, the purposes for the collection of user data or retention period. Regarding the processor, a good example is a cloud provider when it handles data according to the controller's instructions. In more complex ecosystems, different actors may participate in the same data flow which makes responsibility and control harder to understand.

2.2.3 Personal Data and Identifiability

It is possible to give a definition of personal data by using the GDPR definition where personal data are "*any information relating to an identified or identifiable natural person*" [3, Art. 4(1)]. Thanks to this definition, it is possible to understand that personal data means information which refers directly or indirectly to an individual, including both identifying attributes and contextual information related to a particular individual.

Data category	**Examples**
Identity data	Full name, aliases, usernames, account handles, and identification numbers.
Contact data	Email addresses, street addresses, and telephone numbers.
Device and network identifiers	IP addresses, MAC addresses, and persistent device or host identifiers that can be linked to an individual.
Location data	Street addresses, geographical indicators, place of birth, and other information that may reveal an individual's location.
Biometric and image data	Photographs, facial images, fingerprints, retina scans, voice signatures, and other biometric templates.
Activity and usage data	User actions, audit logs, activity records, and other information that can be used to trace or reconstruct an individual's behavior.
Sensitive personal data	Medical, financial, educational, employment, and criminal-history information.

Table 2.1. Main PII categories relevant to consumer platforms, derived from the examples provided in NIST SP 800-122 [2]

An important point of the definition cited above is that in order to be defined personal data there is no need for the data to contain an individual's name in order to fall within the GDPR. Furthermore, NIST SP 800-122 uses the concept of Personally Identifiable Information (PII) to describe information that can lead to trace an individual's identity, and this includes any information that is linked or

linkable to a person [2]. This is useful in platform environments, since identifiability usually comes from combinations of data rather than from a single explicit identifier. In Table 2.1, it is possible to see different examples of PII categories.

2.2.4 Data Sharing as Controlled Processing

Under the GDPR, the term sharing falls into a broader notion of processing, which includes different operations that can be performed on personal data such as collection, storage, use, disclosure, dissemination, alignment, combination, erasure, and destruction [3, Art.4(2)]. This means that when a platform shares data with another user or exposes it through an API or any other kind of action, it performs processing operations. Table 2.2 presents some GDPR provisions that are important in order to understand data sharing as a controlled processing activity.

GDPR reference	**Main concept**	**Relevance for data sharing**
Article 4(2)	Processing	Data sharing is part of processing because it may involve disclosure, transmission, making available, alignment, or combination of personal data.
Article 5(1)	Core principles	Data sharing must respect lawfulness, fairness, transparency, purpose limitation, data minimization, integrity, confidentiality, and accountability.
Article 5(1)(b)	Purpose limitation	Data collected for one purpose should not be reused or shared for incompatible purposes.
Article 25	Data protection by design and by default	Sharing settings, visibility options, access permissions, and third-party connections should be designed to avoid unnecessary exposure by default.
Article 32	Security of processing	Data sharing requires appropriate safeguards, such as access control, authentication, encryption, pseudonymization, logging, and regular testing.
Article 4(5)	Pseudonymization	Pseudonymization can reduce direct identifiability, but it does not remove data from the GDPR if re-identification remains possible.

Table 2.2. GDPR provisions relevant to data sharing [3]

This table shows that the problem is related to how the entire processing chain is governed. Starting from the user interface, sharing appears as a single basic

action such as pressing a button for example, then at platform level, the same action must be stored, associated with metadata, disclosed or processed by external service providers. For this reason, access to personal data should be understood as part of the processing lifecycle. When a data item is collected for a specific reason, its access should be limited to the actors and operations necessary for that function. Access control can also be considered as both a technical and legal means, where the purpose limitation principle, data minimization principle, confidentiality, and accountability are legally required principles that must be complied with. Article 25 of GDPR states that data protection by design and default is mandatory, and EDPB guidelines 4/2019 state that controllers have the responsibility of integrating data protection principles within their processing activities so that, by default, only the necessary data will be processed [3, Art. 25][10]. This means that in a consumer platform, sharing mechanisms should be designed so that unnecessary visibility, excessive permissions, and uncontrolled third-party access are not the default configuration. In order to support this approach, there are also technical safeguards. Article 32 GDPR requires controllers and processors to adopt appropriate measures to the risk, including confidentiality, integrity, availability, resilience, pseudonymization, encryption, and regular testing of safeguards [3, Art. 32]. These measures are necessary because every additional actor or component involved in a data-sharing flow increases the number of possible access points. Authentication, authorization, access logging, encryption, pseudonymization, and limitation of permissions are therefore part of the broader governance of personal data. According to Article 4(5) pseudonymization refers to the processing of personal data in a way that data cannot be attributed to a specific data subject without additional information, provided that such information is kept separately and protected [3, Art. 4(5)]. On the other hand, if re-identification is still possible, then the data will still fall under the jurisdiction of the GDPR. Finally, the use of data should be examined from the perspective of controlled use: in a privacy-aware system where the user can share the data, it should guarantee that there is justified and limited access, and the default settings are privacy-friendly.

2.3 Architecture of Consumer Data Sharing Platforms

It is possible to represent consumer data-sharing platforms through a simplified data-flow architecture. The goal is to identify the main components that perform different actions on personal data. The following representation uses the idea of a data flow diagram, which is used to show how data move through different services. In the context of privacy, it is particularly relevant, as it is possible to see how different actions are performed on data along the way.

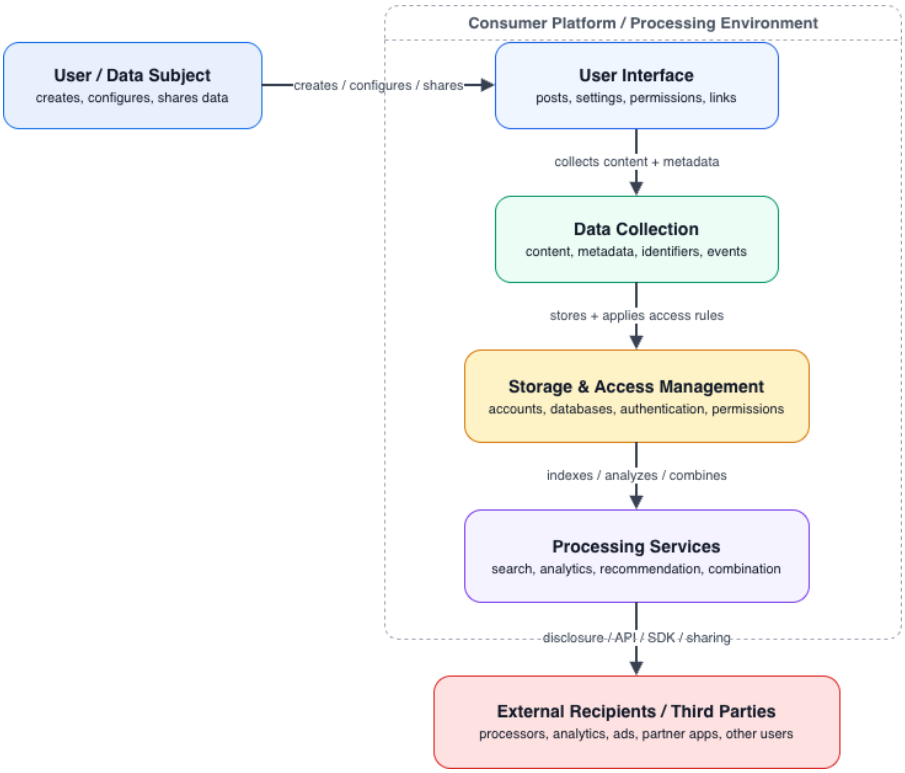

Figure 2.1. Simplified data-flow architecture of a consumer data-sharing platform

At a general level, a consumer platform can be represented through five functional areas:

- **User interface**: the layer through which users create, configure, and share data;
- **Data collection**: the point where the platform receives user-generated data and related contextual information;
- **Storage and access management**: the backend layer responsible for storing data and enforcing access rules;
- **Processing services**: the components that analyze, organize, or reuse data to support platform functionalities;
- **External recipients and third parties**: external actors or services that may receive or process data outside the core platform environment.

This structure is consistent with the idea of a broader data processing ecosystem [6] and is also consistent with the GDPR distinction between controllers, processors, recipients and third parties, because different components of the architecture may correspond to different legal or operational roles [9].

It is possible to see that the user typically sees only an element in the user interface, while at the platform level the action performed by the user in the step

before triggers different operations, for example metadata generation, synchronization, access-control updates and so on. This difference between visible interaction and internal data flow is one of the main reasons why consumer data sharing is difficult to understand and control.

For example, if a user wants to share a document through a cloud platform, the user will see only the creation of a link. But at the architectural level, there are different actions that are performed, starting with the platform that stores the file, then associates it with the account, applies permission rules, logs the event, synchronizes it across devices, and may expose the resource through APIs or connected services.

Finally, this simplified architecture can be used in the following sections for modeling. It allows identifying the main points where attacks can occur, and it also makes clear that data sharing is a process distributed across technical layers and organizational actors.

Chapter 3

Threat Modeling

3.1 Definitions

Threat modeling is defined in [11] as "*the structured process of identifying, analyzing, and prioritizing potential threats to a system*" and this process allows recognizing an attack before its real execution, similarly, in [12] threat modeling is described as a structured and repeatable process which allows one to gain information about the security implemented by a system . There are different steps in this process starting from the modeling of a system from a security perspective, then identifying applicable threats, and finally determining appropriate responses to these threats.

To summarize, a threat model is a structured representation of the information that is related to the security of an application, it gives an overview of the application and its environment through the lens of security. There are four questions that can help to organize threat modeling:

- **What are we working on?**: defines the system, its scope, and its relevant boundaries;
- **What can go wrong?**: it means the identification of threats, weakness, threat actors or any information that can lead to a loss of security;
- **What are we going to do about it?**: this is about mitigations, controls, and risk treatment decisions;
- **Did we do a good job?**: which verifies if the analysis and the responses for it are adequate for the system under consideration.

Threat modeling is a proactive approach which means that it is usually performed during design, before the implementation begins in order to identify the weaknesses before deployment. It can be done during architecture review to reduce uncertainty and improve design decisions earlier in the system lifecycle. It can be done during major changes so when the architecture of the system is modified. But it is also important to do it periodically since threat landscapes evolve and new techniques emerge which means that old methods become outdated. Threat modeling is not simply a testing activity; it is a design and architecture activity. It

helps in the identification of assets, architecture, attack surfaces, threat actors and attack paths before they become operational failures [11] [12].

Finally, as seen before, the goal of threat modeling is to identify the threats to which appropriate countermeasures can be applied; in modern systems, these countermeasures are increasingly configured in an automated way (Section 3.6).

3.2 Attack trees and Attack graphs

In the following section some threat modeling methodologies will be introduced, they show what kind of threats to consider when modeling a system but they don't say much about how the resulting attacks are represented. For this reason, it is important to introduce the concept of attack path which shows the steps that an attacker can do from an initial condition to the compromise of an asset. Of the different types used, two deserve particular mention: attack trees and attack graphs.

Attack trees, introduced by Schneier [13], represent attacks in a hierarchical, goal-oriented form. It is possible to describe them starting from the root, which is the goal that the attacker wants to achieve, and then the rest of the tree represents all the possible ways in which he is able to reach the goal, and this process continues until the leaves represent the attacks. There are two types of nodes: OR nodes that represent the different ways in order to achieve the same goal and there are AND nodes that represent the different steps needed to achieve a goal. So an attack path comes from how the leaves are combined in order to satisfy the root node's objective. An advantage of attack trees is that they can have some attributes, such as cost, likelihood of success, or chances of being detected. These values propagate from the leaves up to the root, which allows a comparison between different attack scenarios to determine which scenario would be cheaper or more probable. Another advantage is related to their simple structure and straightforward representation. The characteristics described until now can lead to one possible limitation for example it is possible to see that attack trees describe a single goal, which means that if there is a system that has several goals it requires several trees.

Now it is possible to introduce attack graphs, which is the other method that allows us to see the possible paths of the attacks. In [14], the nodes represent the various states of the system, or better the privileges of the attacker and the edges represent various atomic exploits or attacks performed by the attacker which take him from one state to another. Every such attack or exploit has certain preconditions and postconditions, where the preconditions of one step must be satisfied by the postconditions of the previous one, which then leads to attack paths, and these are the paths in the attack graph that start at some initial state and end up in a state where the asset under threat is compromised. Thanks to their structure, they are able to represent more than one vulnerability exploited at the same time and with each vulnerability exploited paving the way for others. So, it is important to use them in security analysis as they can be computed and analyzed automatically starting from a model of the system, i.e., its components, their interconnections, and the vulnerabilities known to be present [14]. This allows

answering questions such as whether it is possible to reach a target, or which nodes/edges constitute all the attacks. The problem with attack graphs is related to their scalability, as the number of reachable states may increase exponentially.

To make a brief comparison between the two methodologies just mentioned: the attack tree notation is goal-oriented and ideal for performing qualitative analysis of the process by which one may reach a particular goal; the attack graph formalism is state-oriented and lends itself to automatic analysis of the process by which one reaches multiple states in an entire system. Both formalisms share the notion of the attack path and the possibility that the achievement of one state makes possible another.

3.3 Threat Modeling Methodologies

There are different methodologies for threat modeling, because of this, there is no single universally accepted method that applies to every system, domain, or organizational context. The differences between methodologies are based on the target that they want to analyze for example some methodologies are more design-driven, while others may target the behavior of attackers, privacy issues, or other features. This means that the methodology that will be used depends on what is the goal of the analysis.

3.3.1 STRIDE

STRIDE threat modeling technique is able to map in a systematic way the possible threat categories in a system, this is done during system design and each category represents a violation of a security property. It has been developed by Microsoft as part of its security development practices. The method is design-centric, component-oriented, and classification-based since it is able to group threats into six predefined categories. This reduces ambiguity and helps analysts cover the most common classes of security threats [1].

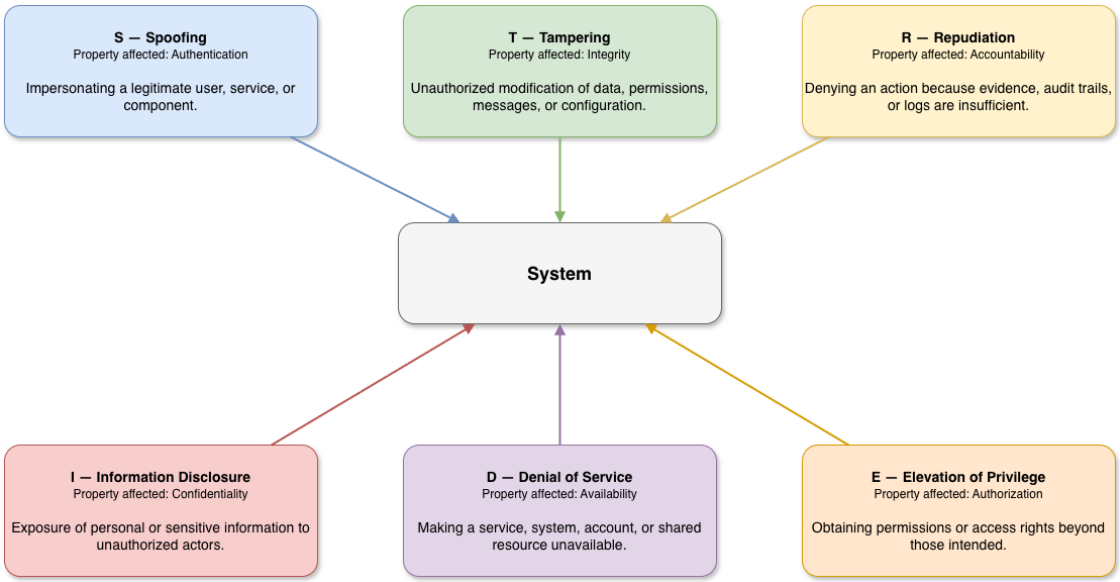

Figure 3.1. STRIDE Threat Model [1]

Advantages of STRIDE include mapping abstract security properties to specific types of threats. For example, if there is authentication in a system, the question posed would be: Is it possible for an attacker to spoof? Where there is any kind of storage or transmission of data, the question posed is: Is it possible for an attacker to tamper or disclose information? If a service must remain available, denial of service must be considered. Another advantage is that STRIDE is particularly useful during the design and architecture review phase. In fact, it can be applied systematically to each component or data flow, asking which of the six threat categories may affect that element. Thanks to this, it is lightweight and practical which means that the analyst does not need to start from a blank page, because the categories provide a ready-made checklist for threat identification [1].

However, STRIDE also has limitations. It is not primarily adversary-driven which means that it does not describe the behavior of the attackers or the way they sequence attacks over time. Another challenge is that the framework does not measure risks quantitatively and does not detect some forms of sophisticated attacks which are a result of interaction among various elements.

STRIDE can be used for design analysis, but if there is the need for modeling real attacker behavior, it is not suggested. For this reason, STRIDE is useful as a first methodological layer, but it is often complemented by other approaches [1].

3.3.2 MITRE ATT&CK

MITRE ATT&CK is not a traditional threat modeling methodology in the same sense as STRIDE. It can be seen as a knowledge base of real-world adversary behavior. In [15] it is possible to consult the official MITRE documentation, which defines ATT&CK as a "*globally-accessible knowledge base of adversary tactics and*

techniques based on real-world observations", this can be used for developing threat models and methodologies in the private sector, government, and cybersecurity community.

Three are the levels which are used by MITRE ATT&CK to organize adversary behavior:

- **Tactics**: the tactical goal of the attacker so what he is trying to achieve;
- **Techniques**: the methods that the attacker uses in order to achieve these goals;
- **Procedures**: which are the implementation of a technique by a specific threat actor.

The main goal of MITRE ATT&CK is to describe the behavior of the attacker instead of describing the vulnerabilities. It documents tactics, techniques and procedures, which helps in understanding how real attackers operate and shows that it can complement design-oriented methods. It is especially useful for:

- mapping realistic attacker techniques to components of the system;
- identifying possible attack paths after a high-level threat has been recognized;
- supporting detection engineering and defensive planning;
- connecting threat modeling with threat intelligence;
- validating whether existing controls cover known adversary behaviors.

There are some challenges, MITRE ATT&CK is large, detailed, and operationally oriented which means that it does not automatically provide a system design analysis or privacy impact, instead the analyst has to decide which tactics and techniques are relevant to the system being analyzed [16].

3.3.3 PASTA

PASTA stands for Process for Attack Simulation and Threat Analysis, is a risk-centric threat modeling methodology with the goal to connect technical threat analysis with business impact. The PASTA framework features a process-centric approach that begins at the business goals level and gradually shifts towards technical analysis, threat analysis, and attack simulation [17].

The main idea behind PASTA is that threat modeling should not remain limited to the identification of abstract threats. Instead, it should explain how realistic attack scenarios may affect the system and what impact those scenarios may have on the organization. Thanks to this, PASTA is mainly used when the goal is to connect technical weaknesses with business consequences, compliance requirements, and risk-based decision-making. This methodology usually has seven stages:

Stage	Name	Purpose
1	Define business objectives	Identify business goals, security objectives, compliance needs, stakeholders, and success criteria.
2	Define technical scope	Establish system boundaries, components, technologies, data flows, and trust boundaries.
3	Application decomposition	Break down the application into modules, assets, data stores, communication channels, dependencies, and integration points.
4	Threat analysis	Identify threat actors, motivations, attack patterns, and threats relevant to the system.
5	Vulnerability and weakness analysis	Identify technical, design, configuration, and operational weaknesses that could enable attacks.
6	Attack modeling and simulation	Build attack scenarios and analyze how an attacker could exploit weaknesses through concrete attack paths.
7	Risk and impact analysis	Evaluate likelihood, impact, and business consequences in order to prioritize mitigations.

Table 3.1. Seven stages of the PASTA methodology

One strength of PASTA is that it is able to integrate security assessment with the organizational context. PASTA does not stop at the existence of a threat; rather it also considers what the threat means for the organization, how it can be leveraged, and the consequences of its exploitation. As such, the PASTA method becomes useful when there is the need to prioritize and rank technical threats based on organizational significance and risk implications. Yet, PASTA is more resource-intensive compared with lighter approaches [18].

3.3.4 LINDDUN

LINDDUN is a threat modeling methodology in which its main focus is on privacy threats, since it is privacy-focused technique, which makes it suited to systems that process personal data. The LINDDUN acronym refers to the types of privacy threats it can identify, and in Table 3.2 the types are shown.

LINDDUN threat type	**Meaning**
Linking	The ability to associate two or more data items, actions, or events with the same individual.
Identifying	The ability to determine the identity of a data subject from available information.
Non-repudiation	The inability of a data subject to deny having performed an action, which may create privacy concerns when accountability becomes excessive or unwanted.
Detecting	The ability to infer whether a data subject is involved in a system, activity, or communication.
Data disclosure	The exposure of personal or sensitive information to unauthorized actors.
Unawareness	The situation in which data subjects are not sufficiently informed about how their data are collected, processed, shared, or used.
Non-compliance	The failure to comply with legal, regulatory, or policy requirements concerning privacy and data protection.

Table 3.2. LINDDUN privacy threat types

An advantage of this technique is that it shifts the analysis from generic security properties to privacy-specific concerns. While other methodologies may classify an event as information disclosure, LINDDUN is able to distinguish which kind of privacy issue occurs based on linking, identifying or the other types of threat for which it was designed to capture. This can be particularly useful in some data-sharing platforms because a system can be secure in terms of authentication, encryption, or access control, but there are still problems from a privacy point of view. As seen in Chapter 2 the user may see only the interface and not how the data is processed behind the scenes. Because of this, LINDDUN is usually applied to system elements related to data such as data flows, data stores.

However, LINDDUN also has limitations. It is privacy-oriented, so it does not fully replace security methodologies such as STRIDE or MITRE ATT&CK. It allows identifying privacy threats, but it is less focused on modeling adversary techniques, malware behavior, infrastructure compromise, or operational attack chains. In order to address this problem, LINDDUN is usually combined with other security-focused methods, these can help identify general security weaknesses or describe adversary behavior, and LINDDUN can address privacy-specific harms [19][20][21].

3.3.5 TAMELESS

TAMELESS stands for Threat & Attack ModEL Smart System and it is a threat modeling and threat analysis tool designed for hybrid smart systems. It is defined as a hybrid system because it takes into account the human, physical and cyber point of view and there are hybrid attacks because they are multi-step attacks which are the combination of the different attack steps that exploit all the dimensions

considered. From an operational point of view, TAMELESS is based on an XSB Prolog interpreter. It represents the system through components, relations, threats, and security assumptions [22].

The input that TAMELESS needs is related to the system to be analyzed such as the components of the system which include all the cyber, physical, and human elements, the threats that can affect those entities, the relation among components and the relation between components and threats and also the security assumptions of the state of the system. After this part, it is possible to perform the threat analysis where the user can query the model in order to retrieve information about the security properties of the system components. This is possible thanks to derivation rules, where TAMELESS is able to derive how threats spread in the system and if, for example, a system component is compromised how it can impact other components.

The framework focuses on deterministic analysis which means that the relations and derived security properties are evaluated using logical reasoning which means that something will be inferred, or nothing will. It relies on static analysis, which means that if there is a system change, it is necessary that the analysis must be run again and finally, the framework also provides graphical support since it is able to generate graphical representations, such as attack graphs in order to visually show the possible attack paths of threats through the system.

The main goal of TAMELESS is to analyze hybrid threats in systems where cyber, physical, and human components interact, which makes it suitable for smart systems which is the reason it was made. A challenge of the framework is that the system needs to be modeled explicitly using components, relations, threats, and assumptions which means that if some relations are missing, it is not able to deduce their implications [22].

3.3.6 Comparative analysis

Methodology	Main focus	Best suited for	Main limitation
STRIDE	Security threat categories: spoofing, tampering, repudiation, information disclosure, denial of service, and elevation of privilege.	Early design review, component analysis, data-flow analysis, and identification of classic cybersecurity threats.	Does not model real attacker behavior in depth and is not specifically designed for privacy harms.
MITRE ATT&CK	Real-world adversary tactics, techniques, and procedures.	Understanding how attackers operate, mapping attack behaviors, supporting detection, and connecting threats to known techniques.	It is not a complete design methodology and must be adapted to the specific system architecture.
PASTA	Risk-centric and impact-oriented threat modeling.	Connecting technical attack scenarios with business impact, prioritization, and risk-based decision-making.	More complex and time-consuming than lightweight approaches such as STRIDE.
LINDDUN	Privacy threat types such as linking, identifying, disclosure, unawareness, and non-compliance.	Analyzing privacy risks in systems that process personal data, especially data flows and user-related information.	Less focused on operational attacker behavior and technical intrusion chains.
TAMELESS	Rule-based threat modeling for hybrid cyber-physical-human systems.	Analyzing smart systems where cyber, physical, and human components interact, using derivation rules and formal reasoning.	Requires an explicit and complete system specification; missing relations limit what the model can infer.

Table 3.3. Comparison of the threat modeling methodologies

3.4 Advantages and Challenges

Looking at the previous section, it is clear that threat modeling is a very useful technique, it is possible to see that it provides a structure through which one may

analyze the possible threats and mitigation strategies needed to ensure that the important asset remains secure. However, also if it is very useful for what concerns the safeguard of an asset, it is also true that by itself it does not guarantee the complete security of the asset. In this section, both advantages and challenges of its usage will be shown below.

Advantages

Some advantages of Threat Modeling, based on [23] [24] [11], include:

- **Identification of a weakness before implementation**: which means that thanks to its proactive approach it is able to recognize design flaws before the implementation. This is important since the issues related to the design such as excessive permissions, violation of trust boundaries, absence of access control mechanisms are difficult to fix after the deployment;
- **Complete view of the system**: since the analysis requires the identification of assets, interfaces, data flows, actors, assumptions, and dependencies. This makes visible some aspects of the system that are hidden or implicit more visible, which allows a better view in complex platforms where several components and third parties interact;
- **Structured reasoning**: a lot of approaches can be used for the analysis as the ones cited in the past section which are STRIDE, LINDDUN, PASTA, MITRE ATT&CK, and TAMELESS. Thanks to them it is possible to have a more objective approach;
- **Clear understanding within the security team**: since the threat model provides a representation of assets, attack paths and mitigation through diagrams tables or classifications allowing the security team to analyze the system under different perspectives. It is good since each person views risks differently;
- **Helps with mitigation decisions**: once a threat is recognized and consequently the weaknesses, it is possible to correlate this information with controls, design improvements, and security requirements. One example in the context of this thesis can be a vulnerability regarding accessing data through a sharing platform may result in security requirements related to implement a more robust authentication or authorization measures;
- **Connection with risk assessment**: because the analysis done will provide the input for a later risk assessment, which means that a good model will make risk assessment more grounded and less abstract.

Challenges

Some challenges of threat modeling, based on [12] [24], include:

- **Scope definition**: it is necessary to take into consideration the full picture of a system without focusing too much on individual adversaries, assets, or techniques since parts of the model are interdependent;
- **Incomplete information**: since the analysis is based on system representation and there is no single representation that is able to capture the full picture of the system because of different points of view, if there is any missing information, it can make the analysis inadequate;
- **Choice of methodology**: each approach has its own strengths and the absence of a standard requires particular attention on the choice of the approaches;
- **Subjectivity**: analysts have to choose which threats are relevant and which mitigations are necessary, but the result of this choice is based on the experience of the analysts, which may lead to incomplete or inconsistent assessments;
- **Maintenance over time**: since the systems evolve, the model must be updated when the architecture changes or the model risks no longer reflecting the complete picture of the threats.

3.5 Process and Scope

The application of threat modeling in this section follows the same four-question structure introduced earlier: What are we working on?, What can go wrong?, What are we going to do about it?, and Did we do a good job?. OWASP presents these questions as an operational process composed of four corresponding steps: scoping the work, determining threats, determining countermeasures and mitigations, and assessing the work [25].

In this section, the four steps are used not as a new theoretical framework, but as a practical structure for organizing the results of the analysis. First, the scope of the scenario is determined using the system boundary, actors, data flow, data store, and trust boundary. Second, the threats are found based on the identification of the elements of Data Flow Diagrams. Third, countermeasures that are appropriate for the identified threats are determined. Finally, an assessment of the findings will be done to identify what is included and excluded from the model and its usefulness to analyze the security and privacy concerns of the scenario.

Threat modeling question	OWASP process step	Use in this analysis
What are we working on?	Scope your work	Define the scenario, system boundary, actors, data stores, data flows, and trust boundaries.
What can go wrong?	Determine threats	Identify security and privacy threats affecting components, flows, stores, and boundaries.
What are we going to do about it?	Determine countermeasures and mitigation	Associate threats with suitable security and privacy controls.
Did we do a good job?	Assess your work	Evaluate completeness, assumptions, limitations, and usefulness of the analysis.

Table 3.4. Mapping of the threat modeling question to the OWASP process step

3.5.1 Example of Threat Model Process

Scenario Description

The scenario discussed in this section represents a social photo sharing process in which the end-user posts a photo using the social media application. The social platform processes the request by linking the uploaded material to the respective user profile, stores the photo, and can forward the chosen information to an outside provider for distribution purposes or any other reason. The chosen scenario is rather basic, and the purpose is to illustrate the basic example of the consumer's data sharing process.

Step 1: Scope Your Work

The first thing that should be done at step 1 is determining the boundaries of the process being analyzed, which in this case is restricted to a regular process of publishing a photo on a social network. The process starts when the user uploads the content and finishes when the social network stores the photo, associates it with the profile of the user and sends certain information to another service, providing information about whether the content has been published. In this scenario it is possible to see one external user, two internal processes in the social network, two databases, and one external service involved in the process. The basic assumptions include the following:

- the user has a valid account on the social platform;
- the user uploads the photo, caption, and their visibility settings at the same time while posting the photo;
- the platform backend performs the entire process of uploading and posting the photo;

- the uploaded photo is associated with the user profile;
- the photo is stored by the platform;
- selected data can be transferred to the third-party service;
- the scope of analysis includes only threats in the context of privacy and information sharing.

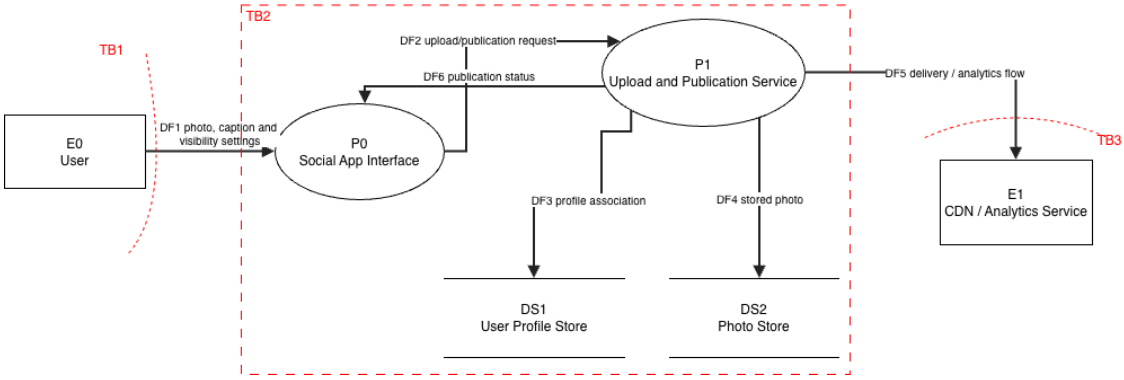

Figure 3.2. Data Flow Diagram of the scenario

Figure 3.2 represents the scenario through a Data Flow Diagram. External entities are indicated with E, processes with P, data stores with DS, trust boundaries with TB, and data flows with DF[26].

DFD Elements The DFD contains the following elements:

Element	Name	Role in the scenario
E0	User	The data subject who submits the photo, caption and visibility settings.
P0	Social App Interface	The user-facing interface through which the publication action is performed.
P1	Upload and Publication Service	The backend service that receives the upload request, applies publication logic, associates the photo with the profile, stores the content and returns the result.
DS1	User Profile Store	Data store containing user profile information and account context.
DS2	Photo Store	Data store where the uploaded photo is stored.
E1	CDN / Analytics Service	External service that may receive selected data for content delivery, analytics or platform optimization
TB1	User Environment	Trust boundary between the user and the platform.
TB2	Social Platform Environment	Trust boundary containing the platform interface, backend service and internal data stores.
TB3	External Service Environment	Trust boundary between the platform and the external service.

Table 3.5. Elements represented in the scenario DFD

Data Flows The numbered data flows are listed below.

Flow	Source	Destination	Description
DF1	E0 - User	P0 - Social App Interface	The user submits the photo, caption and visibility settings as part of the same publication action.
DF2	P0 - Social App Interface	P1 - Upload and Publication Service	The application sends the upload and publication request to the backend service.
DF3	P1 - Upload and Publication Service	DS1 - User Profile Store	The platform associates the uploaded photo with the user profile.
DF4	P1 - Upload and Publication Service	DS2 - Photo Store	The platform stores the uploaded photo.
DF5	P1 - Upload and Publication Service	E1 - CDN / Analytics Service	The platform may send selected data to an external service for content delivery or analytics.
DF6	P1 - Upload and Publication Service	P0 - Social App Interface	The platform returns publication status and updates the user interface.

Table 3.6. Data flows represented in the scenario DFD

Assets The main assets in this scenario are the data items and platform resources involved in the publication process.

Asset	**Type**	**Reason for protection**
Uploaded photo	Content data	May contain personal information, faces, places, objects or contextual details.
Caption	User-generated content	May reveal personal opinions, relationships, context, location or other sensitive information.
Visibility settings	Access-control data	Determine who can view the uploaded photo and under which sharing condition.
User profile	Personal data	Links the uploaded content to the user account and identity context.
Stored photo	Platform data asset	Represents the persistent version of the uploaded content inside the platform.
External service dataset	Shared or derived data	May be used for content delivery or analytics and may create linkability or profiling concerns.

Table 3.7. Main assets in the scenario

Step 2: Determine Threats

Now in the step 2 the main goal is to determine threats and since the scenario involves personal data and user-generated content, the analysis mainly uses LINDDUN privacy threat types[20].

The table below maps selected LINDDUN threat types to the DFD elements and data flows[26].

DFD element / flow	**LINDDUN threat type**	**Threat scenario**	**Possible consequence**
DF1 - Publication input	Identifying	The photo or caption contains elements that allow the user or other people to be identified.	Exposure of identity or contextual information beyond what the user expected.
DF1 - Publication input	Unawareness	The user may not fully understand the effect of the selected visibility settings.	The content may become visible to a wider audience than intended.
DF2 - Upload request	Disclosure of information	The upload request contains personal content and publication parameters that must be protected during transmission and processing.	Personal content or sharing choices may be exposed during the publication flow.
DF3 - Profile association	Linking	The platform links the uploaded photo to the user profile and account context.	The photo becomes part of a broader user profile and may contribute to profiling.
DF4 - Photo storage	Disclosure of information	The stored photo may be accessible beyond the intended audience if access rules are too broad or incorrectly configured.	Unauthorized or excessive exposure of personal content.
DF5 - External service flow	Linking	Selected data sent to the external service may be linked with other service data or repeated interactions.	Creation of behavioral profiles or cross-service tracking.
DF5 - External service flow	Unawareness	The user may not clearly understand that selected data are sent outside the core platform environment.	Reduced transparency and weaker control over downstream processing.
DS1 - User Profile Store	Identifying / Linking	Profile data can connect uploaded content with the user's identity and previous activity.	Stronger identifiability and profiling of the user.
DS2 - Photo Store	Non-compliance	Stored photos may be retained, reused or processed beyond the declared purpose.	Possible violation of purpose limitation, retention limitation or user expectations.

Table 3.8. Mapping of the threats using LINDDUN

Even if the analysis is privacy-oriented, STRIDE can also be applied to some categories. For example, information disclosure overlaps with LINDDUN disclosure threats, but also other STRIDE threats could be applied starting from tampering, which might impact visibility permissions, and elevation of privilege, allowing an attacker to have access to more information than intended. However, the main goal

of this example is to show how privacy-oriented threat modeling can be applied to a data-sharing process.

Step 3: Determine Countermeasures and Mitigation

Once the threats are identified in the previous step, now it is necessary to map each threat to some possible countermeasures. In this scenario, mitigations should not focus only on technical security, but also on transparency, minimization, access limitation and privacy-aware defaults.

Threat area	**Relevant flows / assets**	**Possible mitigation**
Identifiability from photo or caption	DF1, DS2	Provide warnings about visible faces, locations or sensitive context; support private-by-default publication; allow users to review content before publishing.
Unclear visibility settings	DF1	Use clear audience labels, preview of who can see the content, privacy-preserving defaults and confirmation before public publication.
Exposure during upload	DF2	Protect transmission with secure channels, validate publication parameters and avoid including unnecessary data in the request.
Profile linking	DF3, DS1	Limit unnecessary profile association, separate public content from internal identifiers where possible and minimize persistent cross-context identifiers.
Excessive access to stored photos	DF4, DS2	Enforce fine-grained access control, least privilege, access logging, encryption at rest and regular permission review.
External service linkability	DF5, E1	Apply data minimization, pseudonymization, aggregation, purpose limitation, contractual restrictions and third-party assessment.
User unawareness of external processing	DF5	Clearly explain external service flows, provide granular privacy controls and make downstream processing visible to the user.
Non-compliant retention or reuse	DS2	Define retention limits, restrict secondary use, document processing purposes and allow deletion or withdrawal where applicable.

Table 3.9. Possible mitigations for the identified privacy threats

Thanks to these mitigations, it is possible to see that in a data-sharing application the classic methods such as encryption or authentication are not enough for privacy, but they must also be supplemented by privacy-oriented design choices.

Step 4: Assess Your Work

Finally, the last step is to assess whether the analysis is thorough enough and useful. In this example, the main goal of the model was to capture the main privacy-relevant elements during the photo-sharing process and it is possible to see after the analysis that privacy issues may arise, in fact, even during normal platform operations, even a simple photo upload can expose a user to privacy concerns when data access, storage, association, or reuse are not properly controlled.

The example used for the analysis is intentionally simplified and it does not consider all possible platform components, does not quantify risk through likelihood and impact, and does not define a full attacker model. The objective of this case was to illustrate the use of the DFD and LINDDUN threat types in assessing privacy threats in a data-sharing context. It aims to show that data-sharing systems should be analyzed in terms of how data are linked, disclosed, retained, reused, and explained to the user.

3.6 From Threat Modeling to Automated Countermeasures

As seen in the last section, it is possible to understand how effectively threat modeling works by identifying the possible threats in the system and then once they are identified, a separate step is to find the corresponding countermeasures and then apply them to the corresponding threat. Today, there are already some examples in the literature where the configuration of such countermeasures is increasingly carried out in an automated way, thanks to security configuration automation, which aims to generate and maintain the configuration of security mechanisms with limited or no manual intervention [27].

This research direction is currently very active. The general problem of configuration automation, together with its state of the art and open research trends, is discussed in [27]. Several works address the automatic configuration of virtual firewalls in order to minimize the consumption of hardware resources required for their deployment [28, 29], while others target the minimization of energy consumption [30, 31]. Other approaches focus on the automatic reconfiguration of firewalls in response to detected threats [32, 33], on the optimization of firewall configurations so as to avoid anomalies that would otherwise prevent threats from being correctly blocked [34], and on the automatic configuration of Virtual Private Networks [35].

But, as seen in this section there is a lack of threat models and analysis algorithms that allow to link the identified threats and the countermeasures to be applied. These approaches in the literature assume that the relevant threats and security requirements are already known, but they do not specify how these should

be derived in a systematic way from a description of the system. It is precisely in this space that the present work is positioned, as discussed in the following chapter.

Chapter 4

Thesis Objectives

Chapter 2 showed that digital platforms are not limited to a single exchange between a user and a service, while in reality they are composed of different actors that may collect, store, process, share or reuse data in different ways. This leads to several consequences about the usage of a data item where the user sees only the interface-level action while the platform performs different operations in the background.

Chapter 3 instead analyzes the concept of threat modeling which allows us to understand how a system can be threatened. The problem here is that threat modeling approaches are not enough for describing data-sharing ecosystem because they focus on identifying threats within a single system, whereas here the challenge is to follow how data flows, propagates and combines across different services. In fact, the problem is not whether the attacker is able to access a data item, but how it was reached: a data item may become accessible indirectly, through propagation from one item to another, or through the aggregation of several separately shared items into new, more sensitive knowledge.

So, these two chapters introduced the main elements that are needed in order to understand the problem addressed in this thesis. The goal is to create a formal threat model for threat analysis in data-sharing ecosystems. The objective is to find entities, relations and properties, then with the derivation rules it is possible to see how the model derives when a data item becomes accessible and harmful. This thesis focuses on representing data-sharing scenarios in a structured way and deriving privacy or security conditions from explicit rules.

4.1 Thesis Workflow

It is possible to describe the workflow of the thesis in the following four steps:

1. **Analyze the data-sharing problem**: the first objective is to identify different scenarios in order to understand how data items circulate through different platforms and how the attacker is able to aggregate this kind of information to derive new knowledge;

2. **Define the formal model**: the second objective is to define a global formal model, so the main entities, properties and relations needed to represent a data-sharing scenarios;

3. **Implement the model in Prolog**: once the formal model is ready, the third step aims to translate it to a Prolog implementation and then connect it to the web application designed to handle and display its output;

4. **Validate the model through use cases**: finally, the fourth step aims to validate the tool built in the phase before with a set of different data-sharing scenario.

The first step is largely addressed by both Chapter 2 and the example presented in Section 3.5. The final result is a tool that is able to represent how the attacker can reach data items shared through different services and it is possible to see how the aggregation of data from different services allows the attacker to derive new knowledge which leads to the creation of new attack paths.

Chapter 5

Approach and Model

This chapter represents the core contribution of the thesis, here a formal threat model is defined. It is able to reason automatically about which information becomes accessible and harmful. In particular, it describes the system behavior by defining the entities, relations and properties.

5.1 Overview of the Proposed Approach

The following formal model takes inspiration from TAMELESS [22] for what concern the division between entities, relations and properties but the goal is different, as seen in Section 3.3.5 TAMELESS is created for cyber-physical systems, here the goal is to create a formal threat model which performs threat analysis in data-sharing ecosystems. The proposed approach is based on the division of the description of the system from the reasoning about it. The analyst provides a description of the system that the analyst wants to analyze, which are the trusted actors, the data items shared with their associated privacy impact, the trust policies between the actors and the data items, and the possible data items exploitable. Then, thanks to the derivation rules, it is possible to analyze the behavior of the system based on this input.

5.2 Entities

The model defines a set of entities that are involved in the data-sharing ecosystem. It is possible to identify this set of entities:

$$E = \{\, U,\ DC,\ D,\ PI,\ TL,\ AT \,\}$$

In particular:

- U represents actors which include single users, attackers or third parties;
- DC represents data collectors, which are the services through which the data are processed such as platforms, applications, etc.;

- D represents data items;
- PI represents privacy impact levels assigned to the data items;
- TL represents trust levels assigned to actors or data collectors;
- AT represents attack types that may become possible once the data become accessible.

5.3 Relation between entities

The entities defined before are connected by a set of relations that describe how the system is defined so what is the trust level of actors or the privacy impact of a specific data item. These relations are the input of the model defined by the user who wants to use the tool. It is possible to define the following set of relations R:

$$R = \{ \text{assignsImpact, hasTrustLevel, trustPolicy, sharesWith,} \\ \text{sharesWithCollector, givesAccess, aggregableInto, thirdPartyFlow } \}$$

- **assignsImpact** $assignsImpact(D, PI)$: a data item D is assigned a privacy impact level PI by the user;
- **hasTrustLevel** $hasTrustLevel(U, TL)$: actor U has trust level TL defined by the user;
- **trustPolicy** $trustPolicy(TL, PI)$: represents the access control policy defined by the user to define which actors with trust level TL should be allowed to access data with privacy impact PI;
- **sharesWith** $sharesWith(D, DC, U)$: data item D is shared through data collector DC with actor U;
- **sharesWithCollector** $sharesWithCollector(D, DC)$: data item D is shared with data collector DC;
- **givesAccess** $givesAccess(D_1, D_2)$: data item D_1 enables access to data item D_2;
- **aggregableInto** $aggregableInto(\{D_1, \ldots, D_n\}, D_k)$: the set of data items $\{D_1, \ldots, D_n\}$ is sufficient to derive a new data item D_k;
- **thirdPartyFlow** $thirdPartyFlow(DC, D, U)$: the data collector DC makes the data item D available to the actor U.

5.4 Properties

Properties define the status of a data item or an actor as computed by the system. It is possible to define properties in two groups: assumed which are declared by the user/analyst and derived which are automatically derived by the tool from the relations of the model defined before.

5.4.1 Assumed Property

They represent what is known about the system as input, such as which data items are public and which attacks a data item is exposed to.

$$P_a = \{isPublic, exploitableBy\}$$

- **isPublic(D)** $isPublic(D)$: data item D is publicly available;
- **exploitableBy(D,AT)** $exploitableBy(D, AT)$: data item D can be exploited in the context of attack type AT.

5.4.2 Derived Properties

They are computed automatically by the model from the relations and the assumed properties, through the derivation rules. Here is where the value of the approach lies, in particular, canHarm is never stated directly by the analyst, but it comes from the reasoning, revealing harmful conditions that are not visible in the input description.

$$P_d = \{canAccess, trustedFor, canHarm\}$$

- **canAccess** $canAccess(U, D)$: data item D becomes accessible to the actor U;
- **trustedFor** $trustedFor(U, D)$: actor U is trusted for data item D when the trust level assigned to U, according to the trust policy, is sufficient for the privacy impact assigned to D;
- **canHarm(U,D,AT)** $canHarm(U, D, AT)$: actor U can harm data item D in the context of attack type AT.

5.5 Derivation rules

So far, entities, relations and properties have been described and these elements are used in order to describe the system. Derivation rules are where the actual threat analysis takes place, they are able to reason over the description of the system in order to derive which data items become accessible and which actors are in a position to cause harm.

5.5.1 Trust Policy Rules

$$\textbf{assignsImpact}(D, PI) \wedge \textbf{hasTrustLevel}(U, TL) \wedge \textbf{trustPolicy}(TL, PI) \rightarrow \textbf{trustedFor}(U, D)$$
$$\textbf{assignsImpact}(D, PI) \wedge \textbf{hasTrustLevel}(DC, TL) \wedge \textbf{trustPolicy}(TL, PI) \rightarrow \textbf{trustedFor}(DC, D)$$

Table 5.1. Derivation rules for trust

Table 5.1 describes the derivation rules that say which actor or data collector is trusted for a specific data item based on assignsImpact and hasTrustLevel defined by the user, then thanks to the trustPolicy which maps hasTrustLevel to a specific assignsImpact it is possible to derive trustedFor.

5.5.2 Rules for accessibility

$$\textbf{isPublic}(D) \wedge \textbf{actor}(U) \rightarrow \textbf{canAccess}(U, D)$$
$$\textbf{sharesWith}(D, DC, U) \rightarrow \textbf{canAccess}(U, D)$$
$$\textbf{sharesWithCollector}(D, DC) \rightarrow \textbf{canAccess}(DC, D)$$
$$\textbf{sharesWithCollector}(D, DC) \wedge \textbf{thirdPartyFlow}(DC, D, U) \rightarrow \textbf{canAccess}(U, D)$$
$$\textbf{canAccess}(U, D_1) \wedge \textbf{givesAccess}(D_1, D_2) \rightarrow \textbf{canAccess}(U, D_2)$$
$$\textbf{canAccess}(U, D_1) \wedge \cdots \wedge \textbf{canAccess}(U, D_n) \wedge \textbf{aggregableInto}(\{D_1, \ldots, D_n\}, D_k) \rightarrow \textbf{canAccess}(U, D_k)$$

Table 5.2. Derivation rules for accessibility

Table 5.2 defines how a data item can be accessed:

- When it is public;
- When it is shared to a specific user;
- When it is shared with the data collector;
- When it is forwarded to an actor through a third-party flow;

Accessibility is able to propagate itself, thanks to this it is possible to define two other rules:

- A data item can give access to another that becomes accessible;
- The last rule refers to how different data items, when aggregated, lead to new knowledge that is accessible and can later create new attack paths.

5.5.3 Harm Rule

$$\textbf{canAccess}(U, D) \land \textbf{exploitableBy}(D, AT) \land \neg\, \textbf{trustedFor}(U, D) \rightarrow \textbf{canHarm}(U, D, AT)$$

Table 5.3. Derivation rule for harm

Table 5.3 defines the harm rule of the system so that once a data item becomes accessible, it is declared as exploitableBy a specific attack and the actor is not trusted for it, then the actor can harm the data item.

5.6 Application of the model in a use case

In order to illustrate how the rules defined interact, below it is presented a simple example similar to the one cited in Chapter 3, two data items are shared online so they are public. In table 5.4 there is a description of the facts.

Fact	Meaning
trustPolicy$(lowTrust, low)$	A low-trust actor is trusted only for low-impact data.
trustPolicy$(mediumTrust, low)$	A medium-trust actor is trusted for low-impact data.
trustPolicy$(mediumTrust, medium)$	A medium-trust actor is trusted for medium-impact data.
trustPolicy$(highTrust, low)$	A high-trust actor is trusted for low-impact data.
trustPolicy$(highTrust, medium)$	A high-trust actor is trusted for medium-impact data.
trustPolicy$(highTrust, high)$	A high-trust actor is trusted for high-impact data.
assignsImpact$(\textbf{photo}, low)$	A published photo is assigned a low privacy impact.
assignsImpact$(\textbf{checkin}, low)$	A location check-in is assigned a low privacy impact.
assignsImpact$(\textbf{homeArea}, high)$	The inferred home area is assigned a high privacy impact.
hasTrustLevel$(\textbf{attacker}, lowTrust)$	The attacker has a low trust level.
isPublic$(\textbf{photo})$	The photo is publicly available.
isPublic$(\textbf{checkin})$	The check-in is publicly available.
aggregableInto$(\{\textbf{photo}, \textbf{checkin}\}, \textbf{homeArea})$	The combination of the photo and the check-in can reveal the user's home area.
exploitableBy$(\textbf{homeArea}, \textbf{tracking})$	The inferred home area can be exploited in a physical tracking scenario.

Table 5.4. Description of the facts used in the worked example

Based on the facts already described in table 5.4, now it is time to apply the derivation rules to understand the behavior of the system. Starting from the assigned impact given by the user to the data items and the trust for the attacker it is possible to derive the items for which the attacker is trusted for:

$$\textbf{trustedFor}(\textbf{attacker}, \textbf{photo})$$

$$\textbf{trustedFor}(\textbf{attacker}, \textbf{checkin})$$

$$\neg\textbf{trustedFor}(\textbf{attacker}, \textbf{homeArea})$$

Since both the photo and the check-in are public, the public accessibility rule derives that the attacker can access them:

$$\textbf{isPublic}(\textbf{photo}) \wedge \textbf{actor}(\textbf{attacker}) \rightarrow \textbf{canAccess}(\textbf{attacker}, \textbf{photo})$$

$$\textbf{isPublic}(\textbf{checkin}) \land \textbf{actor}(\textbf{attacker}) \rightarrow \textbf{canAccess}(\textbf{attacker}, \textbf{checkin})$$

The aggregation rule can then be applied. Since the same actor can access both input data items, the model derives that the attacker can also access the derived data item:

$$\begin{aligned}&\textbf{canAccess}(\textbf{attacker}, \textbf{photo}) \land \textbf{canAccess}(\textbf{attacker}, \textbf{checkin})\\ &\land \textbf{aggregableInto}(\{\textbf{photo}, \textbf{checkin}\}, \textbf{homeArea})\\ &\rightarrow \textbf{canAccess}(\textbf{attacker}, \textbf{homeArea})\end{aligned}$$

Once the home area becomes accessible to the attacker, it becomes exploitable in the context of a tracking attack, since the attacker is also not trusted for it. Thanks to the harm rule it is possible to derive:

$$\begin{aligned}&\textbf{canAccess}(\textbf{attacker}, \textbf{homeArea}) \land \textbf{exploitableBy}(\textbf{homeArea}, \textbf{tracking})\\ &\land \neg\textbf{trustedFor}(\textbf{attacker}, \textbf{homeArea})\\ &\rightarrow \textbf{canHarm}(\textbf{attacker}, \textbf{homeArea}, \textbf{tracking})\end{aligned}$$

Thanks to this example it is possible to see that the model correctly captures the effect of data sharing. The elements considered are not harmful when they are not aggregated, but once the attacker is able to retrieve information shared from different services and aggregate this information, the new knowledge derived becomes harmful.

Chapter 6

Implementation

In chapter 5 the formal model is defined, in this chapter there is the implementation of it in a working tool. In particular, this chapter describes the implementation of it in detail, first with a description of the architecture, the technologies used and the reason for each, and then walks through the actual code that makes the tool work the Prolog rules that constitute the model, the Python layer that runs them and turns their output into results and a graph, and the web interface that lets a user analyze and build scenarios.

6.1 Architecture Overview

The tool can be seen as three different layers that help to retrieve the output(Figure 6.1):

- **Frontend**: which is implemented in React and renders the attack graph and the results;
- **Backend**: which is implemented in Python, in particular it is used Flask and exposes the analysis over a REST interface and drives the Prolog engine;
- **Prolog engine**: where the formal model is defined and it is executed by SWI-Prolog (swipl), invoked as a subprocess by the backend.

The main core of the tool is the Prolog engine which contains the formal model while the backend and the frontend only orchestrate and present. Thanks to this implementation, the interface can change without affecting the analysis, and the model can be refined without touching the rest of the application.

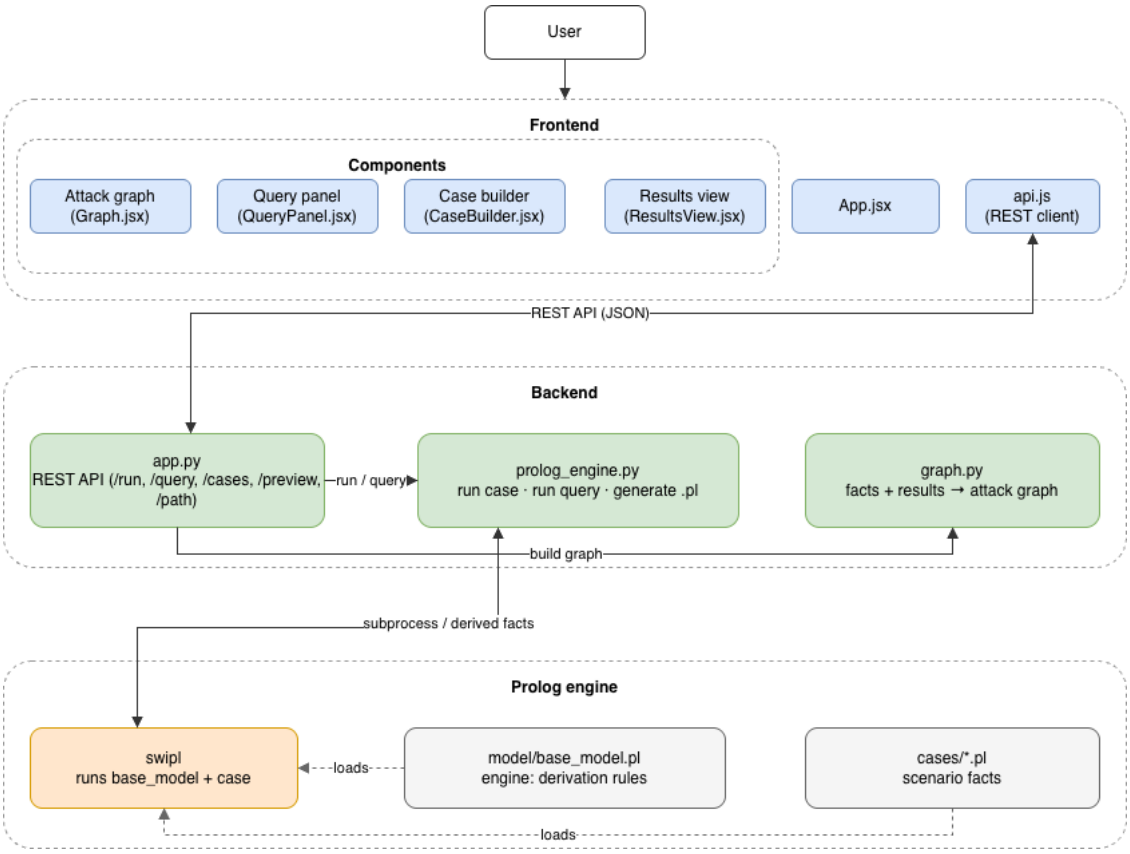

Figure 6.1. High-level architecture of the tool

6.2 Prolog Engine

The core part of the tool is the Prolog engine, this section aims to introduce what Prolog is and finally shows how the model of Chapter 5 is encoded and what each part of the code does.

6.2.1 Prolog

Prolog is the most widely used logic-programming language and the one adopted here. A Prolog program is a set of Horn clauses of two kinds: facts, which assert that a relation holds, and rules, which state that a conclusion holds whenever a conjunction of premises holds. The result is the search for a proof of a query, the engine matches a goal against the head of a clause by unification, making two terms equal by binding their variables, replaces the goal with the clause body, and proceeds recursively, backtracking to alternative clauses when a branch fails.

Prolog uses the closed-world assumption which means that whatever cannot be proved is taken to be false and it realizes negation as failure, where a negated goal succeeds exactly when the attempt to prove it finitely fails.

In the tool the description of a scenario is a set of facts and the derivation rules of the model are Horn clauses, and asking which actor can harm which data item is simply a query whose solutions Prolog enumerates; the closed-world assumption and negation as failure are precisely what the harm rule needs to express that a party is not trusted.

Two practical aspects complete the picture. Because two of the accessibility rules are recursive, the model relies on tabling: the answers of a goal are stored on first computation and reused afterwards, which guarantees termination and computes the least fixpoint of the rules; the negation in the harm rule is stratified, so it is evaluated only after trust and accessibility have been settled. The concrete engine is SWI-Prolog, a mature open-source Prolog system that supports tabling and can be run non-interactively from the command line, so that the backend can drive it as a stateless subprocess.

With these elements, it is possible to show the model in which the rules live in a single scenario-independent file, base_model.pl. It is separated from the use cases that will be mentioned in the validation part and each of them loads this model and lists its facts. The implementation is shown below.

```
% discontiguous are not shown
:- table canAccess/2.
:- table trustedFor/2.
:- table can_harm/3.

actor(A) :- hasTrustLevel(A, _).

trustedFor(U, D) :-
    assignsImpact(D, PI),
    hasTrustLevel(U, TL),
    trustPolicy(TL, PI).

canAccess(U, D) :- isPublic(D), actor(U).
canAccess(U, D) :- sharesWith(D, _DC, U).
canAccess(DC, D) :- sharesWithCollector(D, DC).
canAccess(U, D) :- thirdPartyFlow(_DC, D, U).
canAccess(U, D2) :- canAccess(U, D1), givesAccess(D1, D2).
canAccess(U, Dk) :- aggregableInto(Ds, Dk), all_can_access(U, Ds).

all_can_access(_, []).
all_can_access(U, [D | Rest]) :- canAccess(U, D), all_can_access(U, Rest).

can_harm(U, D, AT) :-
    canAccess(U, D),
    exploitableBy(D, AT),
    \+ trustedFor(U, D).

% Helper for printing results if the model is runned from terminal
```

Listing 6.1. Rules implemented in base_model.pl

The structure of the file follows the concepts that were introduced before. There are three `:- table` directives that mark the derived predicates as tabled, which allow the recursive propagation and aggregation rules to terminate and yield the least fixpoint. Then there is the clause `actor(A) :- hasTrustLevel(A, _)` which

derives the set of actors from the trust assignments and allows the analyst to never declare actors explicitly. Then there is the clause `trustedFor`, which associates the data item's privacy impact with a party's trust level through the policy defined by the user. The six `canAccess` clauses are the ones related to the accessibility of a data item, in order there are: when a data item is public, when it is shared with an actor, when it is shared with a collector, when it is forwarded to a third party, when a data item gives access to another data item and finally when a data item is obtained by aggregation once all its inputs are reachable. The last two clauses are recursive and produce the multi-step access paths that are the whole point of the model. Finally, `can_harm` by referring to the output of the previous properties derives that an actor can harm a data item that it can reach and that is exploitable, but only when it is not trusted for it.

6.3 Backend

The backend connects the reasoning engine of Section 6.2 to the web interface. The main objective is to run the Prolog engine and then turn the output obtained into structured data and a graph and then send these elements to the frontend.

6.3.1 Python

It is a high-level, general-purpose, interpreted programming language known for its readable syntax and its large standard library. It is chosen since it performs well with the operations that are necessary for the orchestration so launching a subprocess, manipulating text and JSON, and wiring a few functions together. In particular Flask is used, which is a lightweight web framework for Python which maps the routes to Python functions and makes it straightforward to return JSON, the text-based format used to exchange data with the frontend.

6.3.2 Running the model

In order to run the model, the backend treats Prolog as a stateless batch computation which means that by looking at Listing 6.2 it copies the base model and writes the case in a temporary directory, runs swipl non-interactively with a goal to execute, captures its standard output, and finally deletes the directory. A timeout prevents a pathological input from blocking the service.

```
def _run(pl_text, goal, timeout=30):
    tmp = tempfile.mkdtemp(prefix="apath_")
    try:
        shutil.copy(MODEL, os.path.join(tmp, "base_model.pl"))
        case = os.path.join(tmp, "case.pl")
        with open(case, "w") as f:
            f.write(pl_text)
        return subprocess.run(
            ["swipl", "-q", "-g", goal, "-t", "halt", "case.pl"],
            cwd=tmp, capture_output=True, text=True, timeout=timeout,
```

```
        )
    finally:
        shutil.rmtree(tmp, ignore_errors=True)
```

Listing 6.2. Running the prolog engine as a subprocess (`prolog_engine.py`)

The string goal passed to swipl using the -g flag in Listing 6.2 is a fixed enumeration goal which is defined once in the backend in Listing 6.3 and its only purpose is to print the derived facts so that Python can read them back. The goal iterates over each derived predicate with forall and prints one line per fact, prefixed by a short tag that identifies its type, and it is possible to see `PUB` for public items, `CAN` for accessibility, `EXP` for exploitability, `TRU` for trust and `HARM` for harm, with the remaining fields separated by tabs. Since every fact is emitted on the same output stream, the tag is what lets the parser decide, line by line, which result a fact belongs to, tagging the lines makes the output trivial to parse and keeps the contract between Prolog and Python explicit.

```
EMIT = (
    "(forall(isPublic(D0),(write('PUB\\t'),write(D0),nl)),"
    "forall(canAccess(A1,D1),
    (write('CAN\\t'),write(A1),write('\\t'),write(D1),nl)),"
    "forall(exploitableBy(D2,A2),
    (write('EXP\\t'),write(D2),write('\\t'),write(A2),nl)),"
    "forall(trustedFor(P3,D3),(write('TRU\\t'),write(P3),write('\\t'),
    write(D3),nl)),"
    "forall(can_harm(A4,D4,T4),(write('HARM\\t'),
    write(A4),write('\\t'),write(D4),write('\\t'),write(T4),nl))),"
    "halt"
)
```

Listing 6.3. EMIT in prolog_engine.py

Once the output is produced, it is then parsed into a structured result by the function _parse, Listing 6.4, and each tag is mapped into a list of records and the set of reachable data items is computed from the accessibility facts.

```
def _parse(out):
    public, can_access, trusted, exploitable, harm = [], [], [], [], []
    for line in out.splitlines():
        parts = line.split("\t")
        tag = parts[0]
        if tag == "PUB": public.append(parts[1])
        elif tag == "CAN": can_access.append({"actor": parts[1], "data_item":
            parts[2]})
        elif tag == "TRU": trusted.append({"principal": parts[1], "data_item":
            parts[2]})
        elif tag == "EXP": exploitable.append({"data_item": parts[1], "attack":
            parts[2]})
        elif tag == "HARM": harm.append({"actor": parts[1], "data_item":
            parts[2], "attack": parts[3]})
    reachable = sorted({c["data_item"] for c in can_access})
    return {"public": public, "can_access": can_access, "reachable": reachable,
           "trusted": trusted, "exploitable": exploitable, "harm": harm}
```

Listing 6.4. Parse function in prolog_engine.py

6.3.3 Queries

The tool also allows the user to pose a single query against the case and see the results. In Listing 6.5 it is possible to see how it is implemented, the goal is first checked against a blacklist of dangerous built-ins in order that the query can only inspect the model and then it is wrapped in a findall guarded by catch, and its solutions are printed and read back.

```
BLOCKED = ("shell", "process_", "open(", "close(", "delete_file",
    "exists_file",
          "consult", "halt", "assert", "retract", "tell", "see", "exec",
          "load_files", "use_module", "read_", "write_canonical")

def query_case_text(pl_text, goal):
    goal = goal.strip().rstrip(".").strip()
    if not goal:
        raise PrologError("empty query")
    if any(tok in goal.lower() for tok in BLOCKED):
        raise PrologError("query contains a disallowed construct")
    wrapped = (
        f"catch((findall(({goal}),({goal}),L),"
        f"(L==[] -> writeln('@@NONE') ; forall(member(S,L),writeln(S)))),"
        f"E,(write('@@ERR '),writeln(E)),halt"
    )
    proc = _run(pl_text, wrapped)
    out = proc.stdout.strip()
    if out.startswith("@@ERR"):
        raise PrologError(out[5:].strip() or "query error")
    if out == "@@NONE" or out == "":
        return {"solutions": [], "truth": False}
    sols = [ln for ln in out.splitlines() if ln and not ln.startswith("@@")]
    return {"solutions": sols, "truth": True}
```

Listing 6.5. Function query_case_text in prolog_engine.py

6.3.4 Generating cases and recovering facts

It is also possible for a user to define a scenario in a form through the interface rather than writing it in Prolog by hand. The function in Listing 6.6 takes as input the structured form and turns it into a Prolog file by emitting one block of facts per relation.

```
def form_to_pl(form, title="Custom case"):
    L = [f"% {title}", "% generated by the tool",
         ":- consult('base_model.pl').", ""]

    def block(comment, items):
        L.append(f"% {comment}")
        L.extend(items)
        L.append("")

    block("privacy impact of each data item",
          [f"assignsImpact({d}, {pi})." for d, pi in form.get("impacts",
              {}).items()])
    block("public data items (assumed: reachable by everyone)",
```

```
        [f"isPublic({d})." for d in form.get("public", [])])
    block("trust level of each actor or collector",
        [f"hasTrustLevel({p}, {tl})." for p, tl in form.get("trust_levels",
            {}).items()])
    block("trust policy",
        [f"trustPolicy({tl}, {pi})." for tl, pi in form.get("trust_policy",
            [])])
    block("attacks each data item enables",
        [f"exploitableBy({e[0]}, {e[1]})." for e in
            form.get("exploitable_by", [])])
    sharing = []
    for s in form.get("shares_with", []):
        sharing.append(f"sharesWith({s[0]}, {s[1]}, {s[2]}).")
    for s in form.get("shares_with_collector", []):
        sharing.append(f"sharesWithCollector({s[0]}, {s[1]}).")
    for tp in form.get("third_party_flow", []):
        sharing.append(f"thirdPartyFlow({tp[0]}, {tp[1]}, {tp[2]}).")
    block("sharing", sharing)
    block("access propagation",
        [f"givesAccess({gg[0]}, {gg[1]})." for gg in form.get("gives_access",
            [])])
    block("aggregation",
        [f"aggregableInto([{', '.join(parts)}], {dk})."
         for parts, dk in form.get("aggregable_into", [])])
    return "\n".join(L).rstrip() + "\n"
```

Listing 6.6. Function form_to_pl in prolog_engine.py

In order to build the graph, the backend needs the structure of a case independently of the derived results. In Listing 6.7 it is possible to see how the prolog_engine.py handles it by extracting the facts directly from the Prolog text with regular expressions, after stripping comments, producing a dictionary of impacts, trust levels, public items, sharing relations, propagation and aggregation.

```
def parse_facts(pl_text):
    t = _strip_comments(pl_text)
    g = lambda p: [m.groups() for m in re.finditer(p, t)]
    agg = []
    for m in re.finditer(r"aggregableInto\(\s*\[([^\]]*)\]\s*,\s*(\w+)\s*\)",
        t):
        parts = [x.strip() for x in m.group(1).split(",") if x.strip()]
        agg.append([parts, m.group(2)])
    return {
        "impacts": {d: i for d, i in
            g(r"assignsImpact\(\s*(\w+)\s*,\s*(\w+)\s*\)")},
        "trust_levels": {p: tl for p, tl in
            g(r"hasTrustLevel\(\s*(\w+)\s*,\s*(\w+)\s*\)")},
        "trust_policy": [list(x) for x in
            g(r"trustPolicy\(\s*(\w+)\s*,\s*(\w+)\s*\)")],
        "public": [x[0] for x in g(r"isPublic\(\s*(\w+)\s*\)")],
        "shares_with": [list(x) for x in
            g(r"sharesWith\(\s*(\w+)\s*,\s*(\w+)\s*,\s*(\w+)\s*\)")],
        "shares_with_collector": [list(x) for x in
            g(r"sharesWithCollector\(\s*(\w+)\s*,\s*(\w+)\s*\)")],
        "gives_access": [list(x) for x in
            g(r"givesAccess\(\s*(\w+)\s*,\s*(\w+)\s*\)")],
        "aggregable_into": agg,
```

```
        "third_party_flow": [list(x) for x in
            g(r"thirdPartyFlow\(\s*(\w+)\s*,\s*(\w+)\s*,\s*(\w+)\s*\)")],
        "exploitable_by": [list(x) for x in
            g(r"exploitableBy\(\s*(\w+)\s*,\s*(\w+)\s*\)")],
    }
```

Listing 6.7. Function parse_facts in prolog_engine.py

6.3.5 Building the attack graph

In order to build the graph the function build_graph is created which combines the structural facts with the derived results into a node link graph. It is possible to see three kinds of nodes the data items which carry their privacy impact, public or harmed flags, parties such as actors or collectors with their trust levels, and attacks and the nodes are created on demand by helper functions. The edges encode aggregation, propagation, exploitability, the derived can access, and the derived can harm.

```
def build_graph(facts, result):
    impacts = facts.get("impacts", {})
    public = set(facts.get("public", []))
    harmed = {h["data_item"] for h in result["harm"]}
    harm_pairs = {(h["actor"], h["data_item"]) for h in result["harm"]}

    collectors = set()
    for s in facts["shares_with"]:
        collectors.add(s[1])
    for s in facts["shares_with_collector"]:
        collectors.add(s[1])
    for t in facts["third_party_flow"]:
        collectors.add(t[0])

    nodes = {}
    edges = []

    def d_node(d):
        nid = f"d:{d}"
        if nid not in nodes:
            nodes[nid] = {"id": nid, "label": d, "kind": "data",
                          "impact": impacts.get(d, "unknown"),
                          "public": d in public, "harmed": d in harmed}
        return nid

    def p_node(p):
        nid = f"p:{p}"
        if nid not in nodes:
            nodes[nid] = {"id": nid, "label": p,
                          "kind": "collector" if p in collectors else "actor",
                          "trust": facts.get("trust_levels", {}).get(p, "")}
        return nid

    def a_node(a):
        nid = f"a:{a}"
        if nid not in nodes:
            nodes[nid] = {"id": nid, "label": a, "kind": "attack"}
```

```
        return nid

    for d in impacts:
        d_node(d)

    for parts, dk in facts["aggregable_into"]:
        for p in parts:
            edges.append({"from": d_node(p), "to": d_node(dk),
                          "label": "aggregates into", "rel": "aggregation"})
    for d1, d2 in facts["gives_access"]:
        edges.append({"from": d_node(d1), "to": d_node(d2),
                      "label": "gives access to", "rel": "gives_access"})

    for d, a in facts["exploitable_by"]:
        edges.append({"from": d_node(d), "to": a_node(a),
                      "label": "exploitable by", "rel": "exploitable"})

    for c in result["can_access"]:
        if (c["actor"], c["data_item"]) in harm_pairs:
            continue
        edges.append({"from": p_node(c["actor"]), "to": d_node(c["data_item"]),
                      "label": "can access", "rel": "can_access"})

    for h in result["harm"]:
        edges.append({"from": p_node(h["actor"]), "to": d_node(h["data_item"]),
                      "label": f"can harm ({h['attack']})", "rel": "harm"})

    return {"nodes": list(nodes.values()), "edges": edges}
```

Listing 6.8. Function build_graph in graph.py

There is also a variant of this function which is build_path_graph that produces the graph of a single query in particular by given an actor and a target data item it walks the sub-graph that justifies the target, adding the harm edges when the pair appears among the derived harms. This allows the interface to show, on demand, a specific attack path behind a query rather than the whole configuration.

6.3.6 REST API

The function analyze handles all the steps, run, recover facts and build the graph and every endpoint relies on this same function, so the routes around it have very little to do. In Table 6.1 there are the main endpoints and in Listing 6.9 the function analyze is reported.

Method	Endpoint	Purpose
GET	/api/cases	List the available cases.
GET	/api/cases/<cid>/run	Run the analysis on a case.
GET	/api/cases/<cid>/source	Retrieve the Prolog source of a case (used for testing and programmatic access).
POST	/api/run	Run raw Prolog sent by the client, without saving it (used for testing).
POST	/api/cases	Create and run a case from form data.
POST	/api/preview	Preview the generated Prolog source.
POST	/api/query	Evaluate an ad-hoc query.
POST	/api/path	Generate the attack-path graph for a query.

Table 6.1. Main REST endpoints

```
def analyze(pl_text):
    result = run_case_text(pl_text) # 1. run the Prolog model
    facts = parse_facts(pl_text) # 2. recover the structural facts
    result["graph"] = build_graph(facts, result) # 3. build the attack graph
    result["facts"] = facts
    return result

@app.get("/api/cases/<cid>/run")
def run_builtin(cid):
    src = read_case(cid)
    if src is None:
        return jsonify({"error": "case not found"}), 404
    try:
        result = analyze(src)
        result["source"] = src
        return jsonify(result)
    except PrologError as e:
        return jsonify({"error": str(e)}), 400
```

Listing 6.9. Analyze function and a REST endpoint in app.py

The backend works as the orchestration layer where for each request it calls the Prolog engine which performs the reasoning part using the given input from the backend and then the backend handles the output and finally delivers the results to the frontend.

6.4 Frontend

The Frontend is the part of the tool that allows the user to interact, in particular, it allows the user to select or build a scenario and then display the attack graph and the results from the JSON returned by the backend. It is a single page web application mainly used for displaying the results of the Prolog engine.

6.4.1 React and Vite

React is a JavaScript library for building user interfaces, its idea is to build user interfaces by using single and reusable components and these maintain their own state and when the state changes the component re-renders itself. In this tool it is used to build the interface by splitting it into independent components, each responsible for one concern for example the view of the attack graph or the results tables. Then Vite which is the build tool and development server and it is used to proxy the request to the backend in order to avoid cross-origin problems and to allow the two tiers to run as separate processes. The interactive attack graph is drawn with vis-network, a JavaScript library for laying out and rendering network diagrams in the browser.

```
import { defineConfig } from "vite";
import react from "@vitejs/plugin-react";

export default defineConfig({
  plugins: [react()],
  server: { proxy: { "/api": "http://127.0.0.1:5050" } },
});
```

Listing 6.10. vite.config.js

6.4.2 Rest client

The requests sent by the frontend to the backend are handled by api.js in which there is one function for each endpoint needed and each returns a parsed JSON response. Because of the proxy, the components call relative URLs and never need to know where the backend lives.

```
const J = (r) => r.json();
const post = (url, body) =>
  fetch(url, { method: "POST", headers: { "Content-Type": "application/json" },
               body: JSON.stringify(body) }).then(J);

export const getCases = () => fetch("/api/cases").then(J);
export const runBuiltin = (id) => fetch(/api/cases/$id/run).then(J);
export const previewPl = (form) => post("/api/preview", { form });
export const createCase = (name, form) => post("/api/cases", { name, form });
export const runQuery = (body) => post("/api/query", body);
export const getPath = (b) => post("/api/path", b);
```

Listing 6.11. api.js

The root component which is App.jsx holds the application state and also orchestrates the two modes of the tool, the visualizing of the attack graphs by looking at the built in cases and the creation of a new one. In order to open a single case there is a single asynchronous call whose JSON result is stored in the state and rendered; the sidebar is produced by mapping over the list of cases.

```
export default function App() {
  const [cases, setCases] = useState([]);
  const [selected, setSelected] = useState(null);
```

```
  const [result, setResult] = useState(null);
  const [busy, setBusy] = useState(false);
  // tab, showSrc omitted

  const refresh = () => getCases().then(setCases).catch(() => setCases([]));
  useEffect(() => { refresh(); }, []);

  const openCase = async (id) => {
    setSelected(id); setResult(null); setBusy(true); setShowSrc(false);
    try { setResult(await runBuiltin(id)); }
    finally { setBusy(false); }
  };
  // onCreate, onPreview omitted

  return (
    // ...
      {tab === "cases" && (
        <div className="layout">
          <aside className="sidebar">
            {cases.map((c) => (
              <button key={c.id} className={case-item $selected === c.id ?
                  "active" :  ""} onClick={() => openCase(c.id)}>
                {c.title}
              </button>
            ))}
            {cases.length === 0 && <p className="muted">No cases (is the
                backend on :5050?)</p>}
          </aside>
          <main className="content">
            {busy && <p className="muted">Running the model...</p>}
            {!busy && !result && <p className="muted">Select a use case to see
                its attack graph.</p>}
            {!busy && result && (
              <>
                <div className="toolbar">
                  <button className="ghost-btn" onClick={() =>
                      setShowSrc(!showSrc)}>
                    {showSrc ? "Hide" : "View"} Prolog source
                  </button>
                </div>
                {showSrc && <pre className="pl-src">{result.source}</pre>}
                <ResultsView result={result} />
              </>
            )}
          </main>
        </div>
      )}
    // tab === "build" omitted
  );
}
```

Listing 6.12. The application shell and the open-case handler in App.jsx

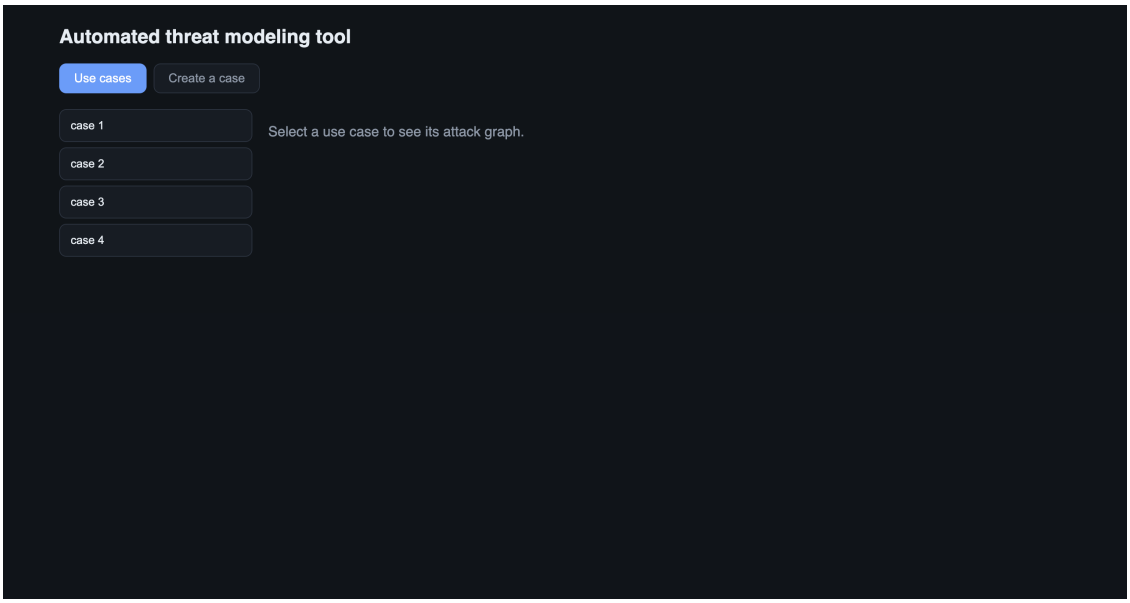

Figure 6.2. Home page with use cases to be selected

6.4.3 Attack graph rendering

Graph.jsx is a component which maps the backend nodes and edges into the attack graph. By looking at the code below, it is possible to see how each node is styled according to its kind and, for data items, its privacy impact, with harmed nodes outlined in red and each edge is styled by its relation type through a small lookup table.

```
import { Network } from "vis-network/standalone";
//...
export default function Graph({ graph }) {
  const ref = useRef(null);

  useEffect(() => {
    if (!ref.current || !graph) return;

    const nodes = graph.nodes.map((n) => {
      const s = nodeStyle(n);
      const sub = nodeSubtitle(n);

      return {
        id: n.id,
        label: $n.label\n($sub),
        shape: "box",
        shapeProperties: { borderRadius: 8 },
        borderWidth: s.borderWidth || 2,
        margin: 12,
        ...s,
      };
    });

    const edges = graph.edges.map((e, i) => {
```

```
    const st = EDGE[e.rel] || EDGE.can_access;

    return {
      id: i,
      from: e.from,
      to: e.to,
      label: e.label,
      arrows: "to",
      color: { color: st.color },
      dashes: st.dashes,
      width: st.width,
    };
  });

  const net = new Network(
    ref.current,
    { nodes, edges },
    {
      nodes: {
        margin: 12,
        font: {
          size: 14,
          color: "#1b1f24",
        },
        widthConstraint: {
          maximum: 230,
        },
      },

      edges: {
        smooth: {
          type: "cubicBezier",
          roundness: 0.45,
        },
        font: {
          size: 11,
          color: "#5b6470",
          strokeWidth: 4,
          strokeColor: "#ffffff",
        },
      },

      physics: {
        solver: "forceAtlas2Based",
        forceAtlas2Based: {
          gravitationalConstant: -140,
          springLength: 230,
          springConstant: 0.045,
          avoidOverlap: 1,
        },
        stabilization: {
          enabled: true,
          iterations: 500,
          updateInterval: 25,
          fit: true,
        },
      },
```

```
        interaction: {
          hover: true,
          dragNodes: true,
          dragView: false,
          zoomView: true,
          navigationButtons: true,
          selectable: true,
        },

        layout: {
          improvedLayout: true,
        },
      }
    );

    net.once("stabilizationIterationsDone", function () {
      net.setOptions({ physics: false });
    });

    return () => net.destroy();
  }, [graph]);

  return <div className="graph" ref={ref} />;
}
```

Listing 6.13. Function Graph in Graph.jsx

6.4.4 Presenting the results and building cases

ResultsView.jsx is a component that assembles the rest of the screen from the JSON returned by the backend which are the derived properties, the attack graph which is rendered by Graph.jsx, the statistics so the results of the harm results for example and the table with the derived harms. It also embeds QueryPanel.jsx which lets the user run a custom query against the current case and offers also different templates of pre-compiled query for example *who can harm a data item?* and then sending them to the /api/query endpoint and another feature is to show the single attack path, it allows the user to choose from the list of actors and the set of data items and then user can select which data item want to see deeply and show the single path related.

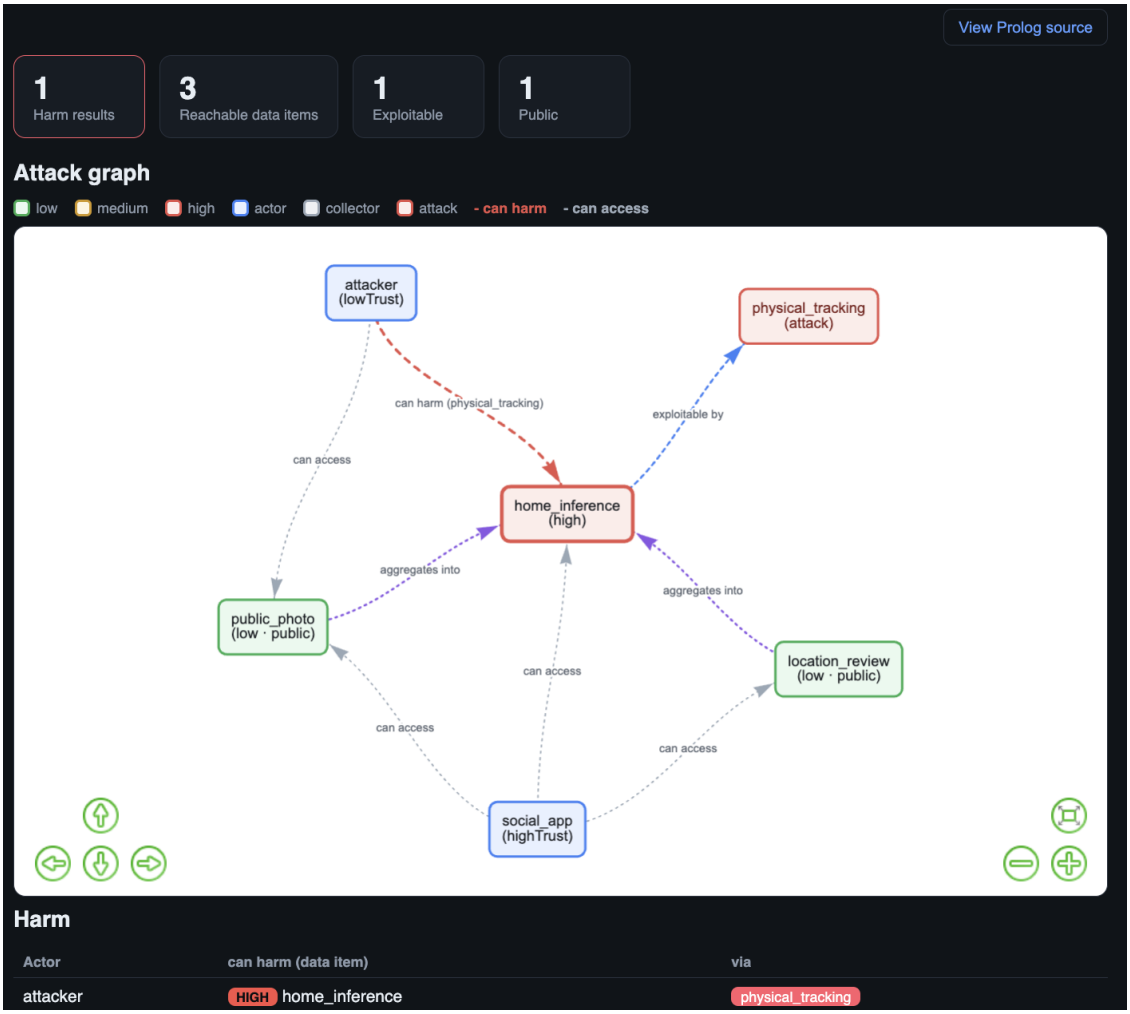

Figure 6.3. Statistics, graph and harm table example

Then there is the CaseBuilder.jsx which is used in the other tab and its main role is to provide the form that the user has to use in order to define a new scenario, then the form can be previewed in Prolog and once the user creates a new scenario it will appear alongside the built-in ones and is handled in exactly the same way.

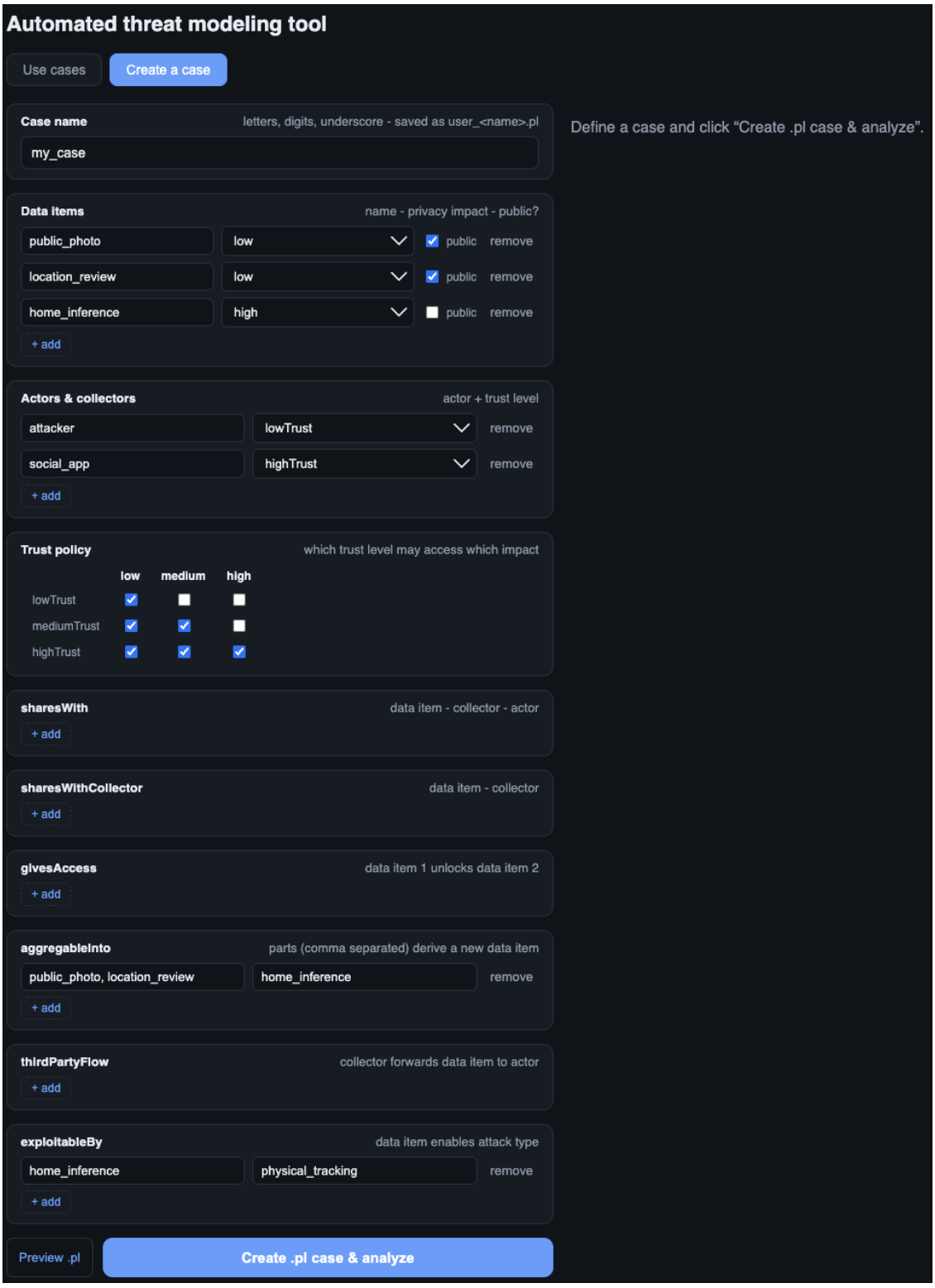

Figure 6.4. Form to create an use case

Chapter 7

Validation

In this chapter, the defined model and the tool created for it will be tested across two different use-cases.

7.1 First use-case

7.1.1 Scenario description

The first use-case is based on a real world case which is the Cambridge Analytica case. This case has been chosen because it is a documented example that shows how easy it is to reconstruct a profile of a person based only on data sharing. According to [36], the starting point of this case is related to the observation that a person's like on Facebook can be used to predict personality traits, so the goal here was to turn this into a voter-profiling and micro-targeting service for political campaigns. The collection starts with the Facebook application developed by the academic Aleksandr Kogan which is the thisisyourdigitallife app. The main goal of the app was to ask the user to take a personality survey, and the app, exploiting a Facebook developer API of the time, harvested not only the data of the people who installed it, but also the profile information of their friends who never interacted with the app at all. This led to acquiring tons of information about people who did not participate in the quiz. The harvested data was then shared with Cambridge Analytica, which used them to generate personality scores and also matched those scores with United States voter records which then led to obtaining rich profiles that were used for voter profiling and creating ad-hoc political advertising.

The model abstracts the combination of Facebook data, personality scores, quiz results, and voter records as a psychographic profile. This abstraction is based on the FTC's description of personality scores being matched with US voter records and were subsequently used for voter profiling and targeted advertising.

7.1.2 Prolog implementation

```
:- consult('base_model.pl').

trustPolicy(lowTrust, low).
trustPolicy(mediumTrust, low).
trustPolicy(mediumTrust, medium).
trustPolicy(highTrust, low).
trustPolicy(highTrust, medium).
trustPolicy(highTrust, high).

hasTrustLevel(facebook, highTrust).
hasTrustLevel(voter_database, mediumTrust).
hasTrustLevel(gsr_app, lowTrust).
hasTrustLevel(cambridge_analytica, lowTrust).

assignsImpact(facebook_likes, low).
assignsImpact(facebook_profile, medium).
assignsImpact(personality_quiz_result, medium).
assignsImpact(voter_record, medium).
assignsImpact(psychographic_profile, high).

sharesWithCollector(facebook_likes, facebook).
sharesWithCollector(facebook_profile, facebook).
sharesWithCollector(personality_quiz_result, gsr_app).
sharesWithCollector(voter_record, voter_database).

givesAccess(personality_quiz_result, facebook_profile).
givesAccess(personality_quiz_result, facebook_likes).

sharesWith(facebook_profile, gsr_app, cambridge_analytica).
sharesWith(facebook_likes, gsr_app, cambridge_analytica).
sharesWith(personality_quiz_result, gsr_app, cambridge_analytica).

sharesWith(voter_record, voter_database, cambridge_analytica).

aggregableInto(
    [facebook_likes, facebook_profile, voter_record, personality_quiz_result],
    psychographic_profile
).

exploitableBy(psychographic_profile, voter_micro_targeting).
exploitableBy(psychographic_profile, targeted_manipulation).
exploitableBy(psychographic_profile, behavioral_influence).
```

Listing 7.1. Implementation of Cambridge analytica case

The implementation defined above shows a deliberately simplified model of a documented case, it aims to test the tool and it abstracts the real events and does not aim to reconstruct their precise technical or legal details.

The model represents the data subject's items which are profile, likes and the quiz result, the platform that collects them which is Facebook and the quiz collector the GSRApp, the sharing of the collected data with Cambridge Analytica, the voter records it was in a position to access, the aggregation of these sources into a single psychographic profile, and the exploitation of that profile by Cambridge Analytica through micro-targeting, targeted manipulation and behavioral influence, the first one is documented in [36] while the other two represent potential threats identified

by the model rather than conducts explicitly described in the FTC's allegations. Some assumptions were made about the voter record because the channel through which they were obtained is not enough detailed in [36], and so they are treated as shared by the voter database. Then the model does not consider the friend of friends multiplication of the harvesting, it captures the flow and composition of the data, not its propagation over the social graph.

The trust levels defined describe the scenario, Facebook is defined as highly trusted because it is the first-party platform to which users directly provided their data, the voter database which contains the voter records is medium trusted and finally the GSRApp and Cambridge Analytica are low trusted since they obtained and processed the data through practices that the FTC described as deceptive, in [36] it is also possible to read that when installing the app there was a message that was explaining to the user that the application would not collect their names or other identifiable information, although it collected persistent Facebook user identifiers and other profile data.

In this use-case that aims to replicate the real case the givesAccess relation abstracts the application authorization process through which participation in the quiz enabled the collection of Facebook profile information and likes, it does not represent the precise technical operation of the Facebook API.

7.1.3 Results obtained

The tool prints the related statistics which are three harm results obtained, five reachable data items, three possible attacks that can happen in the context of the data items and no public data. In this case, the psychographic profile is the harmed data item and it can be exploited in the context of three different attacks.

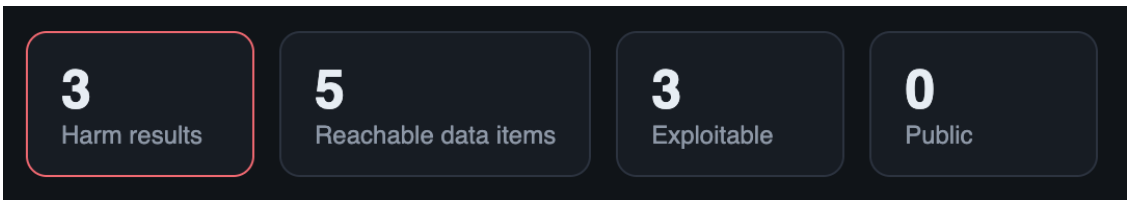

Figure 7.1. Statistics of the first case

In Figure 7.2 there are all the properties including the derived, such as canAccess and trustedFor, and also shows which data item is public or exploitableBy. In this case, it is possible to see that Cambridge Analytica is able to access Facebook profile, Facebook likes, voter record, personality quiz result and psychographic profile. According to the trust policy defined, he should have access only to Facebook likes because it is low-trusted and Facebook likes are low-impact data. However, thanks to the different shared with Cambridge Analytica is able to access to all the data items and it is also able to aggregate them to access psychographic profile which is exploitable. GSRApp can access the personality quiz result Facebook

profile and Facebook likes, however, it is trusted only for Facebook likes. Voter database can access only voter record and it is trusted for it, and Facebook can access only Facebook profile and Facebook likes and Facebook is trusted for them. Then in Figure 7.3 it is possible to see that Cambridge Analytica is able to harm the data.

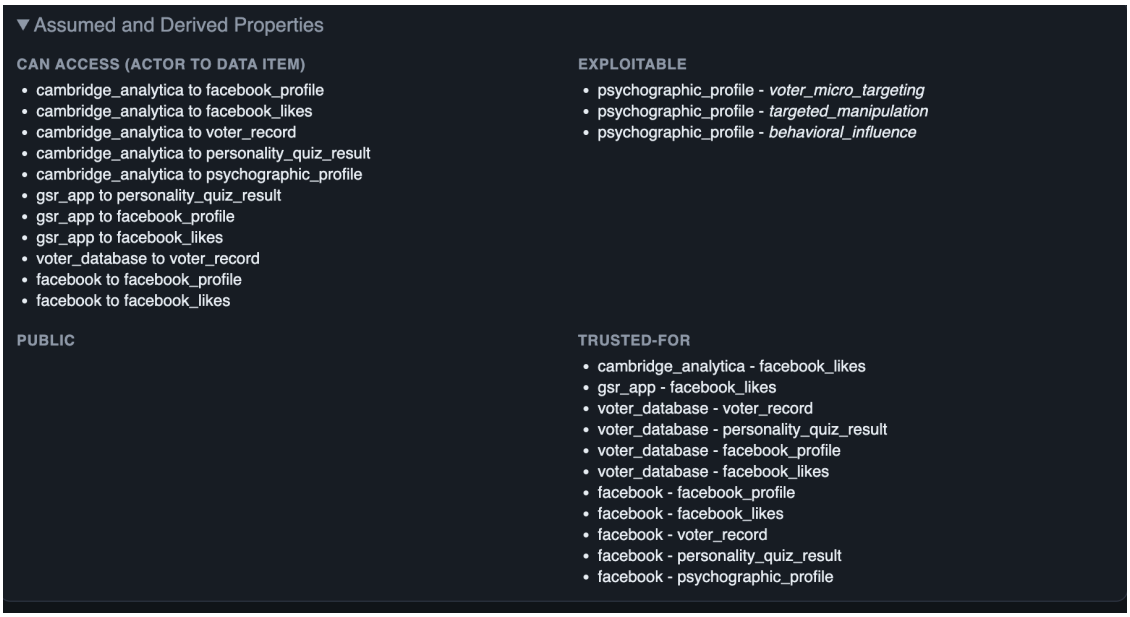

Figure 7.2. Assumed and derived properties of the first case

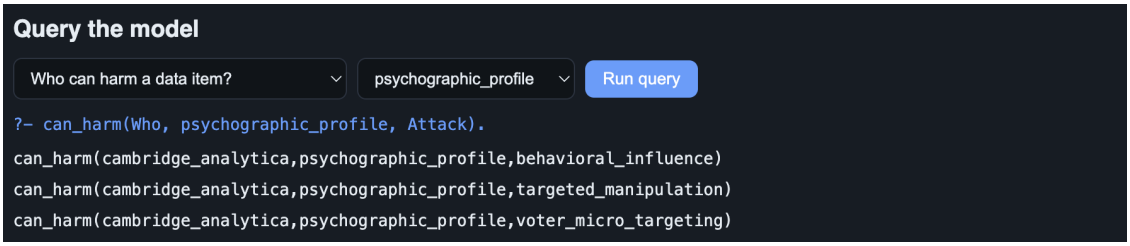

Figure 7.3. Query identifying who can harm a data item

Finally in Figure 7.4 there is the Attack graph output showing all the paths connecting different actors and data items in the use-case. The graph makes a better understanding of the scenario. Looking at the harmful node which exists only as the result of the aggregation, and it becomes reachable only for the actor that can access all of its inputs.

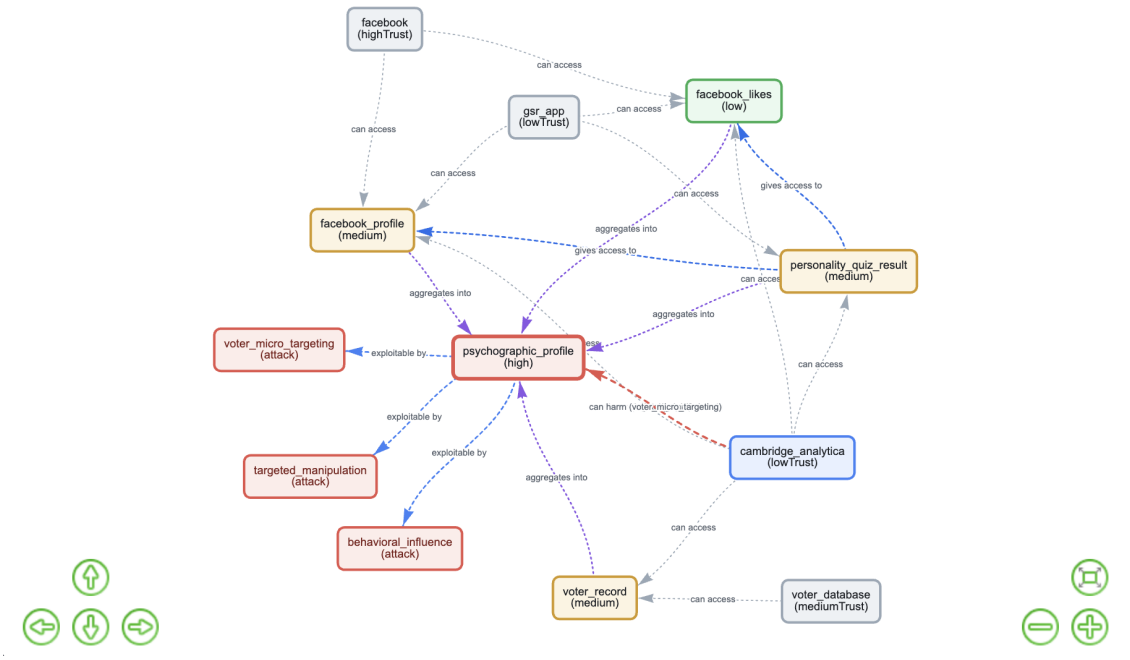

Figure 7.4. Attack graph output

It is also possible to focus only on the path of Cambridge Analytica to the psychographic profile in Figure 7.5 once it obtains access to all the data items of the aggregation.

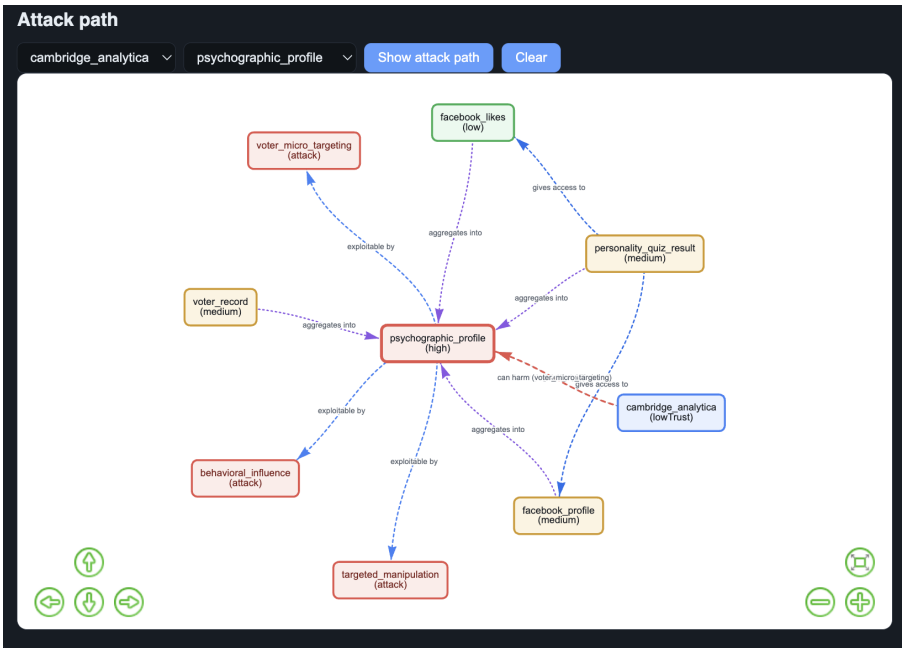

Figure 7.5. Attack path of Cambridge Analytica to psychographic profile

The results obtained match the scenario described before, Cambridge Analytica

is the only one that is able to create harm since it is a low trust actor and has access to the elements that combined give access to the high impact data, the psychographic profile to which it should not have access. The other actors do not cause harm, Facebook is a first party platform which is assigned a high trust level: even though it can access the data items it collects, it is trusted for them, and the harm rule, which requires the actor not to be trusted, does not apply. The GSRApp can access data for which it should not, but it is not able to access the voter record that is needed for the aggregation.

7.2 Second use-case

7.2.1 Scenario description

The goal of the scenario is to represent the normal behavior of the user online, in particular the user interacts with three different platforms which are a social media app, a mapping application and a fitness application and each platform is able to collect different types of location-related information. In this use-case, social media stores a publicly available photo uploaded by the user and also if it is a normal photo, it can contain contextual or location-related information that allows an attacker to discover the user's map check-ins. The maps app stores the user check-ins and the fitness application collects the routes recorded during physical activities. It is supposed that the attacker is able to reach this information through a third party flow. With this example, it is possible to understand how, by combining data from different applications, the attacker can reconstruct the user's daily movements. But not only this, the attacker is able also to infer the user's home location. The daily movement profile may be exploited for behavioral profiling, while the inferred home location may enable physical tracking of the user. The interesting part of the case is that each data item is not particularly sensitive and each data item is managed by the trusted application. The harm in this case comes from the aggregation of data from different contexts. The difference with the first use-case is that it was inspired by a real world documented scenario, while in this case the goal is to test the model on an everyday risk of cross-application data aggregation.

7.2.2 Prolog implementation

```
:- consult('base_model.pl').

trustPolicy(lowTrust, low).
trustPolicy(highTrust, low).
trustPolicy(highTrust, medium).
trustPolicy(highTrust, high).

assignsImpact(social_photo, low).
assignsImpact(map_checkins, low).
assignsImpact(fitness_route, low).
assignsImpact(daily_movements, high).
assignsImpact(home_location, high).
```

```
hasTrustLevel(social_app, highTrust).
hasTrustLevel(maps_app, highTrust).
hasTrustLevel(fitness_app, highTrust).
hasTrustLevel(attacker, lowTrust).

isPublic(social_photo).

sharesWithCollector(social_photo, social_app).
sharesWithCollector(map_checkins, maps_app).
sharesWithCollector(fitness_route, fitness_app).

givesAccess(social_photo, map_checkins).
thirdPartyFlow(fitness_app, fitness_route, attacker).

aggregableInto([map_checkins, fitness_route], daily_movements).
aggregableInto([social_photo, map_checkins, fitness_route], home_location).

exploitableBy(daily_movements, profiling).
exploitableBy(home_location, physical_tracking).
```

Listing 7.2. Implementation of the second use-case

As shown above, the model defines the user's data items so the social photo, map check-ins and the fitness route, and also the trusted data collectors which are the social app, maps app and the fitness app and finally an external attacker. The social photo is public and gives access to the map check-ins. For what concern the fitness route, it reaches the attacker through a third-party flow, that is, the fitness app passes the route it has collected on to an external party. The aggregation of map check-ins and fitness route allows the attacker to retrieve knowledge about daily movements and then combining all the data collected it is possible to access home location. Daily movements and home location are then exploitable by profiling and physical tracking respectively.

7.2.3 Results obtained

In Figure 7.6 the tool prints the related statistics which are two harm results obtained, five reachable items, two possible attacks that can happen in the context of the data items and one public data item.

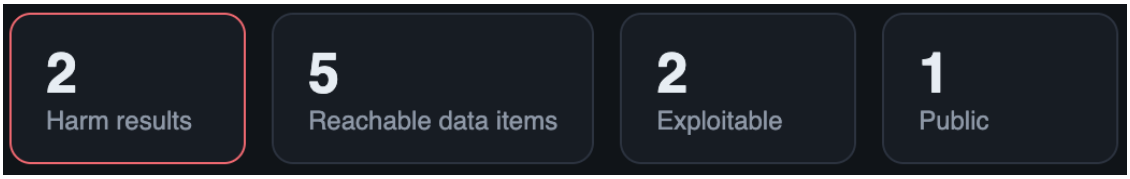

Figure 7.6. Statistics of the second case

In Figure 7.7 it is possible to see that the attacker can access home location, daily movements, social photo, fitness route and map check-ins since some of them

are directly shared with him, or shared photo which is public and the attacker can access it. According to the trust policy, the attacker should access only social photo, fitness route and map check-ins as they are low-impact data items and according to the trust policy defined the attacker can access them. The individual applications can access the data item shared with them plus the social photo since it is public. They are trusted for the data items that they can access because of the trust policy defined by the user. And it is possible to see that since the attacker can access both the home location and the daily movements, which are defined exploitable, and according to the trust policy defined, the attacker is not trusted for them, this means that the attacker can cause harm, Figure 7.8.

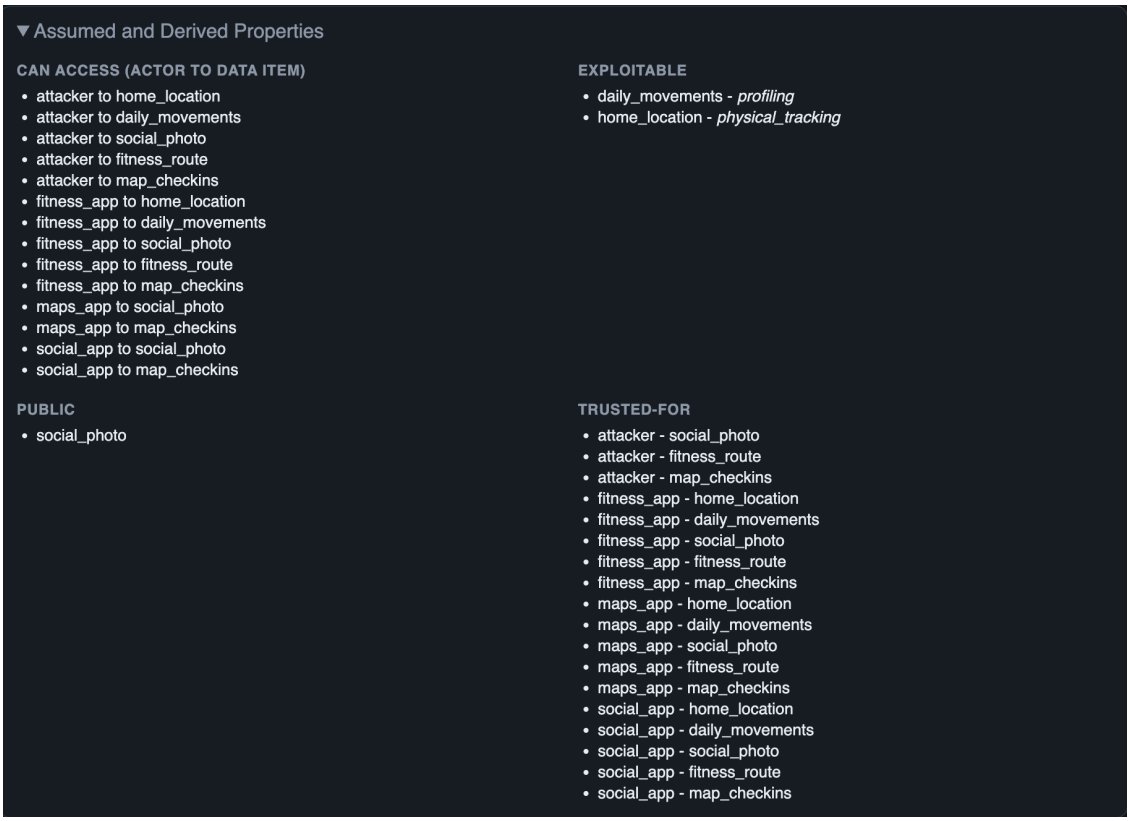

Figure 7.7. Assumed and derived properties of the second case

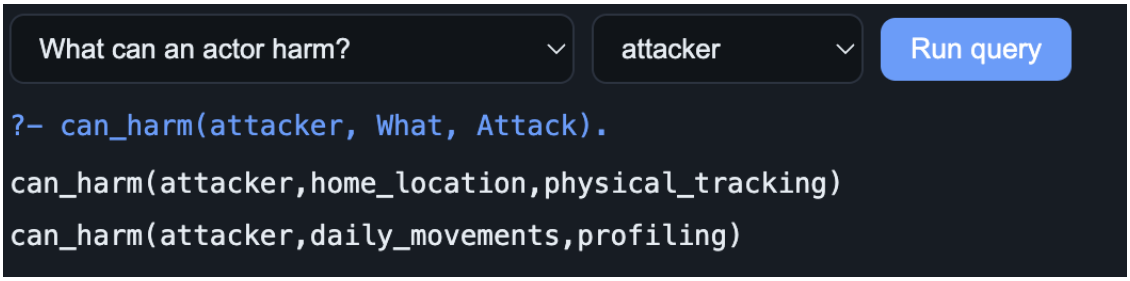

Figure 7.8. Query on which data items an actor can harm

Figure 7.9 shows the attack graph in which it is possible to see the paths through which the attacker can harm both home location and daily movements, via physical tracking and profiling respectively. Everything starts with the public photo which then gives access to the maps check-ins and finally obtains the fitness route through third party flow. Then these three data items are aggregated, and the attacker can reach home location and daily movements. For what concern the data collectors, some of them are able to access and aggregate data items, but they are considered trusted, and the model therefore does not flag their access as harmful. This is one of the main points of the model: the harm is not caused by access in itself, but by access performed by an actor that is not trusted for the data item involved. The two applications involved, the social app and the maps app, are able to combine the data they hold, however since they are trusted for those items, their access is not reported as harmful. Only the external attacker, which is not trusted and reaches the high-impact items through aggregation, produces harm.

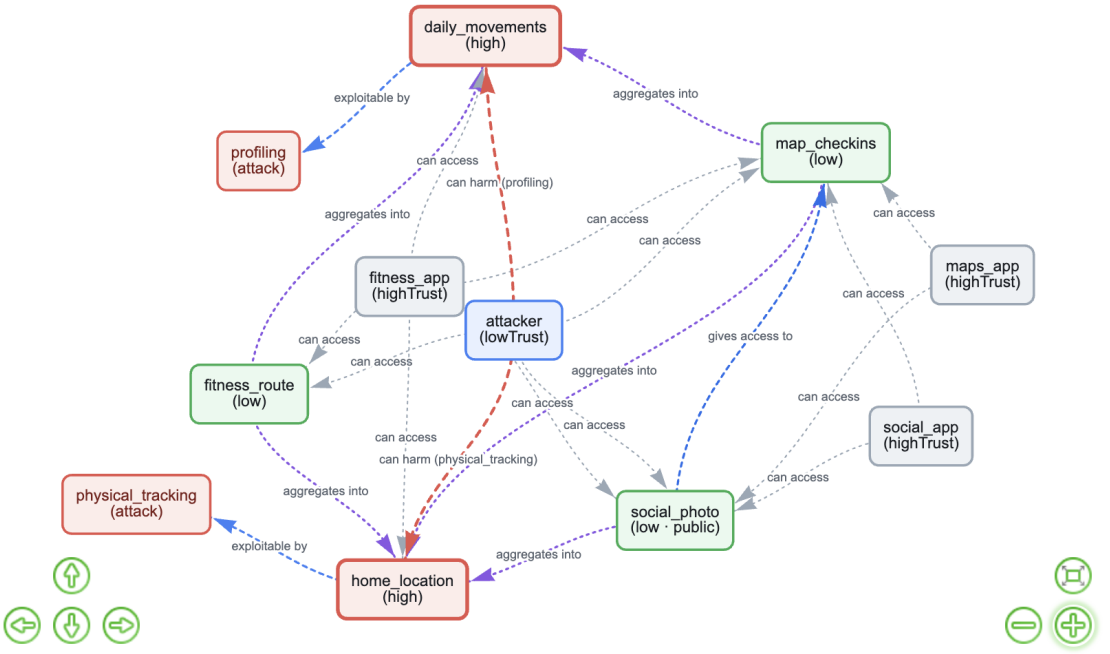

Figure 7.9. Attack graph output

Finally, it is also possible to see the single paths of the attacker to daily movements and home location in Figures 7.10 and 7.11 respectively.

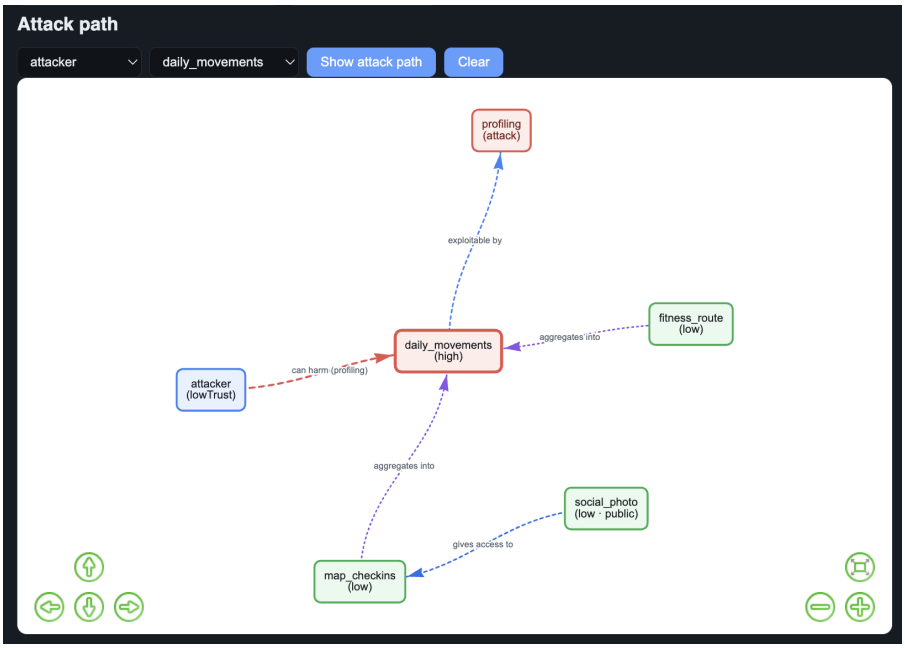

Figure 7.10. Daily movement path

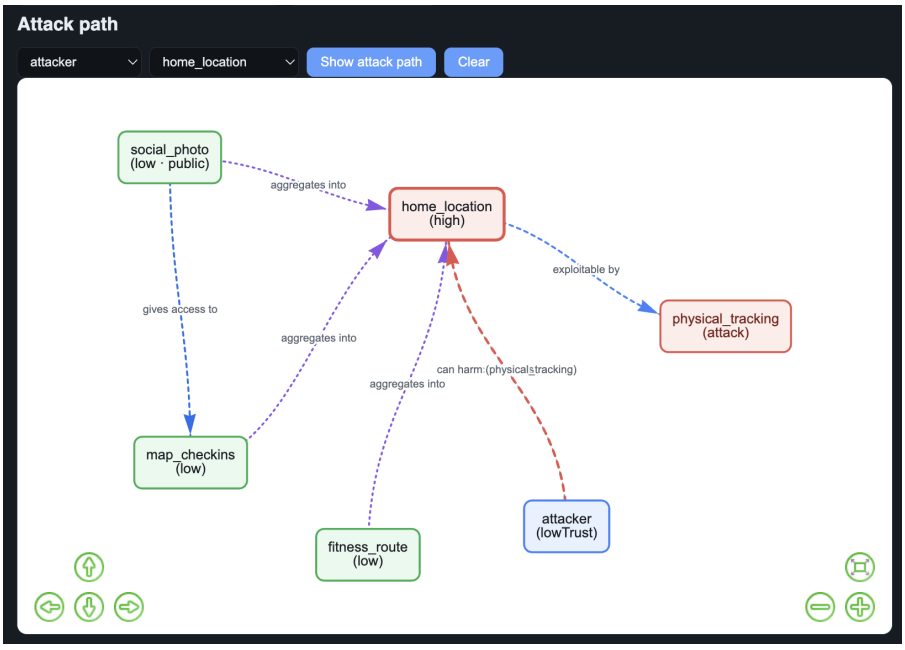

Figure 7.11. Home location path

The results confirm the goal of the scenario so that a single piece of data by itself can be safe, but when it is combined with others it can lead to deriving new knowledge and open doors for the attacker and expose the user to harm.

Chapter 8

Conclusions

In this thesis a new approach, based on threat modeling, was defined to reason about the threats that arise from data sharing, through the definition of a formal threat model. The proposed model allows users to map their own data-sharing system starting from where they have shared their data. This was made possible by the initial definition of a set of entities, relations and properties, which together provide a global view of the data-sharing ecosystem. In particular, the model shows how sharing information across different platforms, however secure those platforms may be individually, can still lead to harm. For example, when a compromise allows an attacker to obtain a data item they should not be authorized to access, or when a data item is shared with the attacker, or maybe because of user unawareness, the attacker can collect data that, once aggregated, yields new knowledge that they did not previously possess. This derived knowledge, which was not initially present, enables new attacks that before were not possible. Thanks to this, it is possible to demonstrate the goal of the thesis that the risk does not come from any single data item or any single platform, but from the aggregation of data from different contexts. The model is then implemented in Prolog, which allows one to perform the reasoning part and then it is assisted by the implementation of a web application whose frontend and backend are used to drive the analysis and to present its output. The tool does not allow only to visualize the possible attack paths, but also to inspect single derived properties and to query the model directly, for example to ask which actors can harm a given data item and also build a new case through a form. The tool was finally tested and evaluated using two different use cases: the first about Cambridge Analytica which is a real documented world case and the second which represents an everyday scenario of cross-application data aggregation. Thanks to the results obtained from them it is possible to see the effectiveness of the approach by showing how data aggregated from different services lead to the derivation of new knowledge that the attacker can exploit.

8.1 Limitations and Future Works

The model is created in order to inspect the privacy aspect of the user, but this means that other security properties such as integrity and availability are not taken into account when the model was built. In fact, there is the possibility that a piece

of shared data is not only disclosed, but also tampered with or made unavailable, and these dimensions are currently not captured. In order to make the model complete and to allow for it to capture also these scenarios, it is possible to add new predicates and rules that describe how a shared or derived data item can be altered or corrupted, this would make the actual model more powerful and able to capture also integrity violations or availability events.

Another possible limitation is related about the fact that the user must know in advance how a derived data item can be attacked which means that he has to know which attack a given data item enables. It is also true that the user can provide a generic attack type but the model would be considerably more powerful if it is added a classifier which is able to find for each type of data the attacks to which they can be exploited. Thanks to this improvement there will be the reduction of manual effort and make the analysis less dependent on the user's expertise.

Further works can be related also about the notion of trust. In the actual model, the user provides manually the trust level of each part and the trust policy is fixed beforehand. In order to improve also this, future works can include the probabilistic treatment of trust, in which an actor is trusted for a given data item only with a certain likelihood estimated, for example, from its past behavior, its category, or known incidents so that the analysis produces a graded notion of risk.

Finally, also the graphical interface, although it is already functional, can be improved. An example is to add the possibility of editing or deleting an existing built-in use case directly from the interface, exporting the generated attack graph and the results as a report, and make the visualization of large graphs more clear in order to make complex scenarios more readable.

Bibliography

[1] V. Kumar, "What is the stride threat model? beginner's guide - 2026," Practical DevSecOps, updated 3 July 2026. [Online]. Available: https://www.practical-devsecops.com/what-is-stride-threat-model/. Accessed: 10-07-2026.

[2] E. McCallister, T. Grance, and K. Scarfone, "Guide to protecting the confidentiality of personally identifiable information (PII)," Special Publication 800-122, National Institute of Standards and Technology, Tech. Rep., 2010. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-122.pdf

[3] European Parliament, Council of the European Union, "Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation)," 2016, [Online]. Available: https://eur-lex.europa.eu/eli/reg/2016/679/oj. Accessed: 10-07-2026.

[4] European Commission, "A european strategy for data," 2020, https://digital-strategy.ec.europa.eu/en/policies/strategy-data. Accessed: 10-07-2026.

[5] OECD, "Recommendation of the council concerning guidelines governing the protection of privacy and transborder flows of personal data," 2013, [Online]. Available: https://legalinstruments.oecd.org/public/doc/114/114.en.pdf. Accessed: 10-07-2026.

[6] National Institute of Standards and Technology, "Nist privacy framework: A tool for improving privacy through enterprise risk management, version 1.0," National Institute of Standards and Technology, Tech. Rep., 2020. [Online]. Available: https://doi.org/10.6028/NIST.CSWP.01162020

[7] OECD, "Enhancing access to and sharing of data: Reconciling risks and benefits for data re-use across societies," OECD Publishing, Tech. Rep., 2019. [Online]. Available: https://doi.org/10.1787/276aaca8-en

[8] Article 29 Data Protection Working Party, "Opinion 05/2014 on anonymisation techniques," Tech. Rep. WP216, adopted on 10 April 2014. [Online]. Available: https://ec.europa.eu/justice/article-29/documentation/opinion-recommendation/files/2014/wp216_en.pdf.

[9] European Data Protection Board, "Guidelines 07/2020 on the concepts of controller and processor in the gdpr," European Data Protection Board, Tech. Rep., 2021. [Online]. Available: https://www.edpb.europa.eu/our-work-tools/our-documents/guidelines/guidelines-072020-concepts-controller-and-processor-gdpr_en

[10] European Data Protection Board (EDPB), "Guidelines 4/2019

on article 25: Data protection by design and by default," European Data Protection Board, Tech. Rep., 2020. [Online]. Available: https://www.edpb.europa.eu/documents/guideline/guidelines-42019-on-article-25-data-protection-by-design-and-by-default_en

[11] Palo Alto Networks, "What is threat modeling?" [Online]. Available: https://www.paloaltonetworks.com/cyberpedia/threat-modeling. Accessed: 10-07-2026.

[12] OWASP Cheat Sheet Series, "Threat modeling cheat sheet," [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Threat_Modeling_Cheat_Sheet.html. Accessed: 10-07-2026.

[13] B. Schneier, "Attack trees," *Dr. Dobb's Journal*, vol. 24, no. 12, pp. 21–29, 1999.

[14] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing, "Automated generation and analysis of attack graphs," in *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, 2002, pp. 273–284. [Online]. Available: https://doi.org/10.1109/SECPRI.2002.1004377

[15] MITRE, "Mitre att&ck," [Online]. Available: https://attack.mitre.org/. Accessed: 10-07-2026.

[16] B. E. Strom, A. Applebaum, D. P. Miller, K. C. Nickles, A. G. Pennington, and C. B. Thomas, "MITRE ATT&CK: Design and philosophy," Mar. 2020, [Online]. Available: https://attack.mitre.org/docs/ATTACK_Design_and_Philosophy_March_2020.pdf. Accessed: 10-07-2026.

[17] Everpure Data, "What is the pasta threat model?" Everpure Data, [Online]. Available: https://www.everpuredata.com/knowledge/pasta-threat-modeling.html. Accessed: 10-07-2026.

[18] ThreatModeler, "PASTA threat methodology," [Online]. Available: https://threatmodeler.com/glossary/pasta-threat-methodology/ Accessed: 10-07-2026.

[19] DistriNet Research Unit, KU Leuven, "Linddun privacy threat modeling," [Online]. Available: https://linddun.org/. Accessed: 10-07-2026.

[20] ——, "Privacy threat types," [Online]. Available: https://linddun.org/threat-types/. Accessed: 10-07-2026.

[21] ——, "Why use linddun?" [Online]. Available: https://linddun.org/linddun/whyuselinddun/. Accessed: 10-07-2026.

[22] F. Valenza, E. Karafili, R. V. Steiner, and E. C. Lupu, "A hybrid threat model for smart systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 5, pp. 4403–4417, 2023.

[23] Victoria Drake, "Threat modeling," [Online]. Available: https://owasp.org/www-community/Threat_Modeling. Accessed: 10-07-2026.

[24] Threat Modeling Manifesto Authors, "Threat modeling manifesto," [Online]. Available: https://www.threatmodelingmanifesto.org/. Accessed: 10-07-2026.

[25] Larry Conklin, "Threat modeling process," [Online]. Available: https://owasp.org/www-community/Threat_Modeling_Process. Accessed: 10-07-2026.

[26] L. Sion and W. Joosen, "LINDDUN PRO privacy threat modeling tutorial," Department of Computer Science, KU Leuven, Tech. Rep., Apr. 2023, version 0.1. [Online]. Available: https://downloads.linddun.org/tutorials/pro/latest/tutorial.pdf.

[27] D. Bringhenti, G. Marchetto, R. Sisto, and F. Valenza, "Automation for network security configuration: State of the art and research trends," *ACM Computing Surveys*, vol. 56, no. 3, pp. 1–37, 2024. [Online]. Available: https://doi.org/10.1145/3616401

[28] D. Bringhenti, G. Marchetto, R. Sisto, F. Valenza, and J. Yusupov, "Automated firewall configuration in virtual networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 2, pp. 1559–1576, 2023. [Online]. Available: https://doi.org/10.1109/TDSC.2022.3160293

[29] D. Bringhenti, S. Bussa, R. Sisto, and F. Valenza, "A two-fold traffic flow model for network security management," *IEEE Transactions on Network and Service Management*, vol. 21, no. 4, pp. 3740–3758, 2024. [Online]. Available: https://doi.org/10.1109/TNSM.2024.3407159

[30] D. Bringhenti and F. Valenza, "Greenshield: Optimizing firewall configuration for sustainable networks," *IEEE Transactions on Network and Service Management*, vol. 21, no. 6, pp. 6909–6923, 2024. [Online]. Available: https://doi.org/10.1109/TNSM.2024.3452150

[31] ——, "Greenshield-e2c: A sustainable energy-aware firewall configuration mechanism for edge-to-cloud continuum," *Computer Networks*, vol. 282, p. 112279, 2026. [Online]. Available: https://doi.org/10.1016/j.comnet.2026.112279

[32] F. Pizzato, D. Bringhenti, R. Sisto, and F. Valenza, "Automatic and optimized firewall reconfiguration," in *NOMS 2024-2024 IEEE Network Operations and Management Symposium*, Seoul, Republic of Korea, May 2024, pp. 1–9. [Online]. Available: https://doi.org/10.1109/NOMS59830.2024.10575212

[33] D. Bringhenti, F. Pizzato, R. Sisto, and F. Valenza, "Autonomous attack mitigation through firewall reconfiguration," *International Journal of Network Management*, vol. 35, no. 1, p. e2307, 2025. [Online]. Available: https://doi.org/10.1002/nem.2307

[34] D. Bringhenti, S. Bussa, R. Sisto, and F. Valenza, "Atomizing firewall policies for anomaly analysis and resolution," *IEEE Transactions on Dependable and Secure Computing*, vol. 22, no. 3, pp. 2308–2325, 2025. [Online]. Available: https://doi.org/10.1109/TDSC.2024.3495230

[35] D. Bringhenti, R. Sisto, and F. Valenza, "Automating VPN configuration in computer networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 22, no. 1, pp. 561–578, 2025. [Online]. Available: https://doi.org/10.1109/TDSC.2024.3409073

[36] L. Fair, "FTC sues Cambridge Analytica for deceptive claims about consumers' personal information," U.S. Federal Trade Commission, Business Blog, Jul. 2019, [Online]. Available: https://www.ftc.gov/business-guidance/blog/2019/07/ftc-sues-cambridge-analytica-deceptive-claims-about-consumers-personal-information. Accessed: 10-07-2026.