Universiteit Leiden
The Netherlands

Opleiding Informatica

Implementing a storage solution facilitating future analysis of ADTs generated by the ADT Web App

Fenna Deutz

Supervisors:
Nathan Schiele & Dr. Olga Gadyatskaya

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

26/06/2024

Abstract

Cybersecurity is playing a constantly increasing role for companies all around the world. One of the tools conducive to securing new systems and improving the security of existing ones is the attack-defense tree (ADT). However, they are not well-used outside academic circles. One of the reasons for this could be the lack of usable applications for creating and modifying attack-defense trees. As a result of ongoing research into this area the ADT Web App was developed which aspires to bridge the gap in usage of attack-defense trees between academia and commercial companies. This thesis aims to implement a storage solution which facilitates further analysis of attack-defense trees generated by the ADT Web App. With the implementation of this new feature this thesis hopes to contribute to ongoing and future research into attack-defense trees and their usability.

Contents

1 Introduction

In the last decade, we have seen an increase in the quantity and gravity of security incidents. Ranging from theft of personal data [Mad] to breaking into computer infrastructures [For]. Defending against attacks is difficult without an adequate risk analysis of the security situation of one's company or organization. This is made worse by the widespread use of third party software, cloud applications and internal applications.

Many of the applications and systems used remain susceptible to security breaches. They can be attacked by criminals that operate within the cyberspace for their crimes. These criminals exploit weaknesses in digital systems known as vulnerabilities. However, there is also an increase in attention for cybersecurity. Several sources on this can be found, focusing on policy [whi], focusing on financial damage [Mor], and focusing on awareness [Cen]. This underscores the imperative to make new systems as secure as possible and to enhance the security of existing systems.

Several methodologies and guidelines exist to enhance the risk analysis. These include standards for the process (e.g. ISO/IEC 27005 [iso]), methods (NIST SP800-30 [fST]) and modeling techniques (such as attack trees [Sch99]). These methodologies provide a framework for the risk analysis process, they support stakeholder interaction and labeling threats.

An improvement of the attack trees are the so-called attack-defense trees (ADT). These upgraded trees include also the defender and therefore also the interaction between attacker and defender. ADTs are used to make a visual representation of an attack-defense scenario. They can identify where potential vulnerabilities lie in case of an attack and how to mitigate them [KMRS11]. Due to the intuitive representation ADTs can also be used by people without extensive cybersecurity knowledge. This makes the ADT a potential tool that can connect academia and corporations. While there are many studies that propose ADT extensions and applications [DZ13][BLN+23][ABP+20], we do not see many industry reports or academic studies about how attack trees are used in industry.

Research attempting to answer the question why ADTs have not reached corporations yet is currently underway. A direction is that in order to overcome this separation between academia and the commercial sector we must increase the usability of ADTs, leading to a more widespread use of ADTs in industry.
A collection of ADTs would be useful for researchers that want to analyze the usage of ADTs. Centralized storage of user-generated ADTs is a method to build this collection. To our knowledge there are no tools that facilitate this kind of storage. As stated above, the objective of the collection of ADTs is to facilitate research into the usage of ADTs. Based on this research the design of the ADT tools could be improved accordingly so that they might reach more users. The current thesis attempts to advance this research into usage of ADTs by investigating the requirements of a centralized storage solution for ADTs generated by a wide group of users.

Furthermore, following these requirements we have developed a possible implementation of the storage. For the implementation, we followed the state of the art in the fields of Systems Engineering (SE) and Requirements Engineering (RE). Using frameworks provided in those fields we traversed four developmental stages to reach a product that satisfied all parties involved. This product consisted of a storage solution for the aforementioned data-set, which was then integrated into an existing tool that allows the creation and modification of ADTs.

1.1 Research goal

The objective of this thesis is to investigate the requirements of a storage solution in order to advance research into usage of ADTs. This will lead to a growing dataset of ADTs to be used in further research or projects. The research question of this thesis is therefore:

> *How to best implement a storage solution which facilitates future analysis of ADTs generated by the ADT Web App?*

Our research question can be divided in two parts. The first being *How to best implement a storage solution...* and the second part being *which facilitates future analysis of ADTs generated by the ADT Web App?*. For the first part we will generate requirements for the storage solution.
The second part is what makes this thesis of practical use to the research group. By creating a storage solution that facilitates further analysis we provide them with a dataset of ADTs that might aid in their research. Section 5.2 will connect these two parts of the research question together by making a trade-off between the requirements defined and the solutions presented in the literature.

1.2 Thesis overview

The current chapter 1 introduces the research goal of the thesis. Chapter 2 contains a background into attack-defense trees and its rules, the context of this thesis and information on the SE and RE standards. This is followed by chapter 3 on some related work. After this chapter 4 describes the possible solutions for XML storage with their advantages and disadvantages provided by the literature. Chapter 5 describes the methodology which includes the entirety of the process that went into accomplishing the research goal. The process is divided into four phases, each of them described in their own section. Chapter 6 contains the result of the thesis, namely the deliverable presented to the supervisor. In chapter 7 we will present per requirement what was done to satisfy it. Finally, chapter 8 contains the conclusion of this thesis and suggestions for future work.

2 Background

2.1 ADT

Schneier coined the term *attack tree* (AT) to name the tree structure used to "describe security of systems and subsystems" where the root node corresponds to the attacker's goal and the structure corresponds to possible attack routes to achieve the goal [Sch99]. Once these ATs gained popularity there was need for a formal interpretation, which was provided by Mauw and Oostdijk [MO06].

Considering the limitation of ATs that they are not able to catch the interaction between attacks and defenses, Kordy et al introduced ADTs as an extension of ATs. In their introductory paper ADTs are described as "a graphical representation of possible measures an attacker might take in order to attack a system and the defenses that a defender can employ to protect the system." [KMRS11]
To ensure the generality and validity of ADTs there are syntactic rules that need to be followed. Below a number of them are described[KMRS11] :

- An ADT has a tree structure with labeled nodes.
- A node is either an attack node or a defense node.
- A node can have one or more children of the same type but only one of the opposite type.
- Refinements (children) are either of type OR, or type AND.
  An OR refinement means that to achieve the goal of the parent node it is sufficient that at least one of the goals of its children are achieved.
  A node refined with the AND type has its goal achieved only when all of its children's goals are achieved.
- Attack nodes are represented by circles, defense nodes by rectangles and refinements (children) by edges between nodes.

Figure 1 displays an example of an ADT. Given that the root node is categorised as an attack node, the goal of this ADT is to attack a Bank Account. A plausible attack-defence scenario is as follows. The attacker can achieve this goal via an ATM or (since the refinement is type OR) via the internet.
Should the attacker choose to attack via the ATM, he/she must obtain both the PIN and the Card of the victim. The PIN can be obtained by either Eavesdrop or Find Note. However, Find Note can be defended by Memorize, eliminating a note to be found. Still, an attacker can deploy a counterattack by using Force.

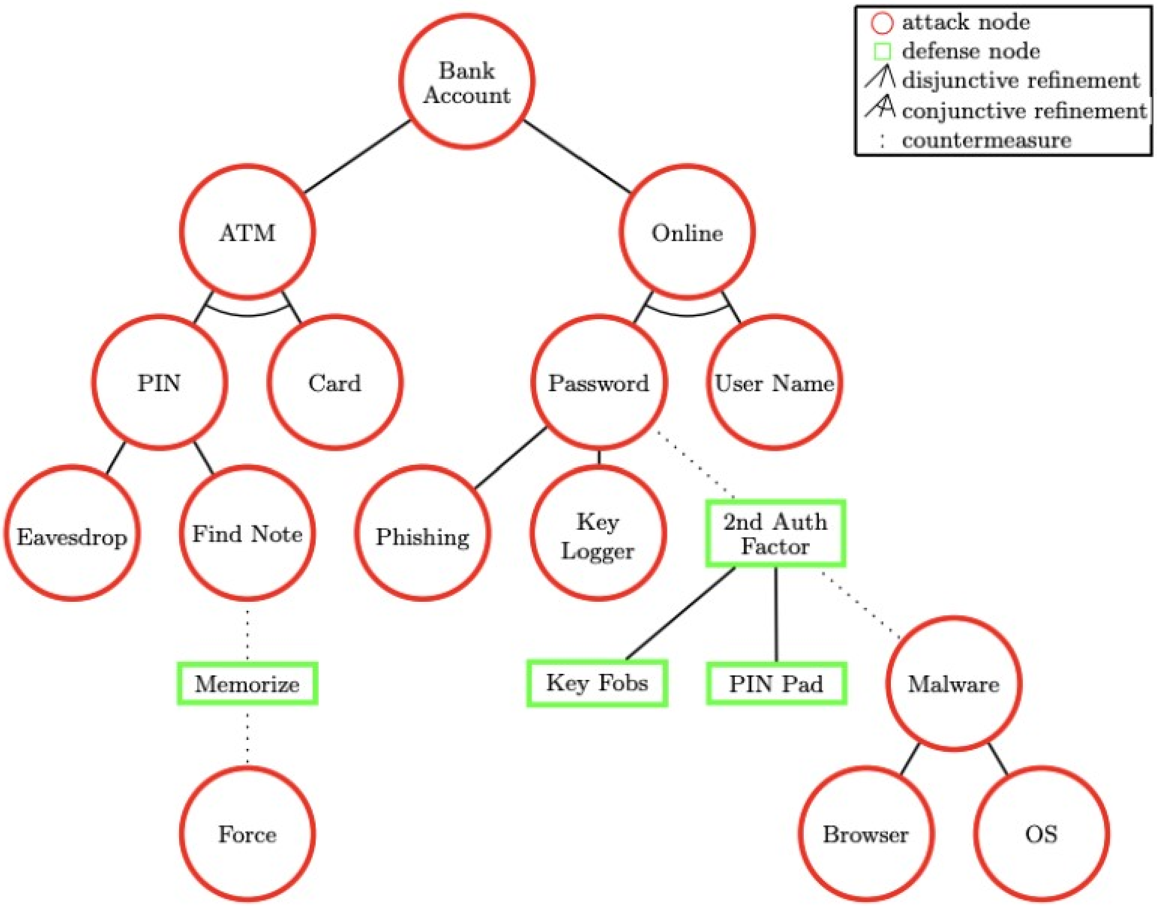


Figure 1: An example of an ADT [KMRS11]

We believe that due to their versatility ADTs could be used in a wide range of areas. They could be used by anyone seeking to make their company, product or service safer and more robust. Since ADTs are quite intuitive to read and create, they are very suitable for non-technical users such as managers, consultants and similar professionals to begin understanding the process that is system security [Sch99].

2.2 Context of this thesis

This thesis is part of broader research into ADTs. Research attempting to answer the question why ADTs are currently used more in academia than in industry is underway. A direction of this research is that in order to overcome this separation between academia and the commercial sector we must increase the usability of ADTs, leading to a more widespread use of ADTs in industry. To have a centralised collection of ADTs is useful for researchers who want to analyse the usage of ADTs.

Based on research conducted on the collection of ADTs, the design of the ADT tools could be improved accordingly so that they might reach more users. The current thesis attempts to advance this research into usage of ADTs by investigating the requirements of a centralised storage solution for ADTs generated by a wide group of users.

This study was conducted in a research group that is active in this area of research. They have developed the ADT Web App in order to have a more user-friendly application which might overcome the separation. The ADT Web App is a user-friendly tool that lets a user create and modify ADTs to visualise attack-defence scenarios in the browser. One of the main requirements of this tool was usability; there is no need for downloading or installing software as it runs in the browser. To simplify the creation of ADTs compared to tools such as ADTool, [KKMS13] the ADT Web App supports ADTLang [Ore]. This is a domain-specific language used for generating valid ADTs, where validity is described such as but not limited to the aspects listed in section 2.1.
The user interface of the ADT Web App was designed keeping in mind principles from the field of human-computer interaction and optimised for usability [Moh]. Figure 2 shows the current state of the ADT Web App.

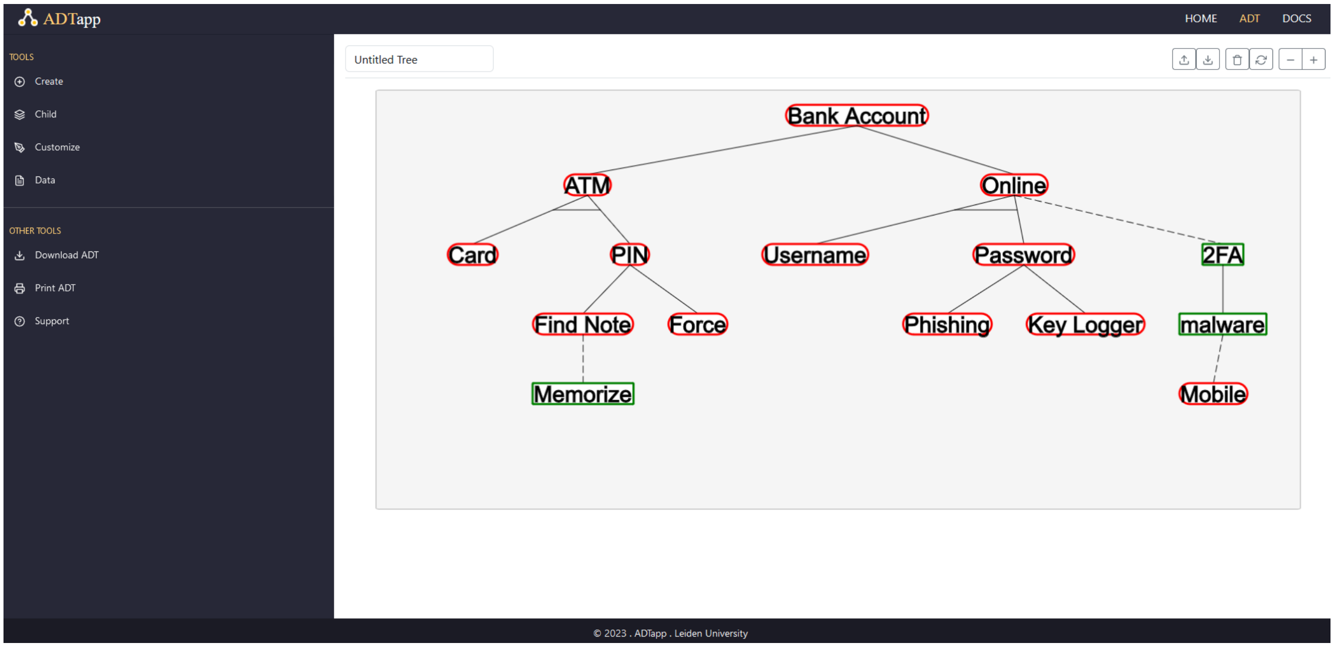


Figure 2: The ADT Web App as it currently is. (https://nschiele.github.io/ADT-Web-App/index.html)

The ADT Web App is operational while the research group continues to expand and improve it. As part of the improvements, this thesis will contribute to ongoing research by providing a storage solution which will contain a (growing) data-set of ADTs generated by the ADT Web App. The storage will be located on the servers within LIACS, so that other researchers can have access to it for analysis. Another potential future feature for the ADTs in the storage is to function as a training set for some model and/or artificial intelligence.

2.3 SE and RE Standards

A major part of this thesis is the design and development of a working storage solution for the ADTs. We have chosen to use a SE approach, which has been the subject of rigorous research.

Although Systems Engineering (SE) is generally used for large and complex systems, its processes can be applied equally well to a small sized product such as the storage solution this thesis describes. SE is a framework of descriptions for describing ”the life cycle of systems created by humans, defining a set of processes and associated terminology from an engineering viewpoint” [Sta23]. SE foundations were laid by conceptualising concepts of systems in engineering and science to general systems methodology. Much groundwork was done by for example Yourdon, Wymore and Klir [You89] [Wym93] [Kli91]. Yourdon's focus was on structured systems analysis, specifically the integration of traditional methods within modern technology. Wymore concerned himself with the application of theory to the design of actual systems. The book written by Klir describes aspects of systems science which he defines as ”...a science whose domain of inquiry consists of those properties of systems and associated problems that emanate from the general notion of systemhood”.

The SE approach is formally defined on a high level in the currently applicable ISO/IEC/IEEE 15288:2023 standard for Systems and Software Engineering [Sta23]. On a more practical level than the generic standard, several organisations have published handbooks containing guidance material for this methodology. For this thesis we have made use of the state-of-the-art handbook of the International Council on System Engineering (INCOSE). Since the inception of the ISO standard in 2002, it is ”formally recognised as a preferred mechanism to establish agreement for the creation of products...” [INC23]. Considering that the deliverable of this thesis is in fact a product we found this standard best to follow. We are not aware of any formal alternatives to this standard, but one could imagine for example that the work for this thesis would immediately start with coding if one does not adhere to the ISO standard.

For the ADT storage solution, we have followed four stages that will be discussed in section 5. These stages are in line with the guidance referenced:

1. Requirements elicitation; including interaction with stakeholders defining the needs and constraints.

2. Finding a solution; starting from the state-of-the-art and including the rejection of unsuccessful alternatives.

3. Developing a prototype solution; focusing on a end-to-end implementation.

4. Implementation of the solution; an obvious last step before the users can store their ADTs.

As mentioned before, we also make use of Requirements Engineering (RE) processes. The Stakeholder Needs and Requirements Definition Process is described extensively in the INCOSE Handbook [INC23]. This state-of-the-art process uses a number of inputs like collecting the source documents (with the current status), project constraints, and stakeholders. The activities of this process included defining stakeholder needs, transforming these needs into requirements. The output is amongst others a set of requirements presented by the stakeholder, listed in section 5.1.

For the inputs, we reviewed the current state of the ADT Web App. The main stakeholders of the project are clear: the thesis supervisors and the LIACS data server managers. Furthermore we discussed project constraints with the main stakeholders; these were taken into account when making the requirements. The core activities of this requirements engineering process were mainly in the form of several meetings with the main stakeholders. As also stated in the cited literature, it is important for the quality and usability of the product to capture the motivation behind the requirements. To ensure this, we explicitly addressed the motivation for requirements during our meetings.
The output of this process is a list of requirements as shown below in section 5.1. This set of requirements is the basis for the next stages.

3 Related Work

This section will discuss research related to this thesis. The deliverable of this research is custom made for the ADT Web App. The ADT Web App is developed in the context of the issue of the gap between academia and the commercial sector in the use of ADTs. At the background of this are considerations in the field of Human Computer Interactions (HCI) referring to usability. Furthermore, there are other tools that facilitate the creation and manipulation of ADTs.

3.1 HCI

In his early book Schneiderman dives into the "specifically human factors that need to be considered in designing interactive computer systems" [SW87]. An important practical guideline described in this book are the eight golden rules. They are still relevant today, and for example used to analyse systems [FFD^{+}22].

Gong et al. conducted an extensive study into HCI theory in their article [GQZ18]. Considering the theoretical aspects they propose "reasonable and practical design strategies, establishes relatively complete and systematic software interface design models, and makes rewarding suggestions on the future of software interface design". They recognise three categories on the usability of a design; Visibility, Usability and Interaction. This interface design model was used in the development of the user interface of the ADT Web App [Moh].

3.2 Tools

There are several tools available for users to work with ADTs. Some are standalone applications while others function as a separate model within an environment. However, while local storage is often available, none of these tools provide a centralised storage solution.
One such tool for creating and manipulating ADTs is ADTool [KKMS13]. ADTool was developed by Kordy et al. and is downloadable from their website for free. The tool is able to perform a quantitative analysis on ADTs which "means that a user is able to answer questions such as: What are the costs of an attack, what is the minimal skill level required for the attacker, how long does it take to implement all necessary defences or who is the winner of the considered attack–defence scenario, and many others"[KS12]. This tool allows for saving and loading of ADTs to the local file system but there is no feature that enables a user to upload their tree to a secure server.

Another tool designed by Kordy et al. is SPTool. This tool aims to extend the ADTs with the SAND refinement that "allow the modeling of sequences of actions" [KKvdB16]. Storage on local file system is available to users, but again there is no storage available on a server.

Yathuvaran et al. developed AT-AT [AYL] which allows users to develop ADTs and analyse them. This tool also does not enable users to save their tree to a server. Local storage of trees is available.

The tool ATSyRa was introduced by Pinchinat et al. and ”provides advanced editors to specify high-level descriptions of a system, high-level actions to structure the tree, and ways to interactively refine the synthesis.” [PAV15]. Its focus is more on military usage (at the time of the tool paper). While it allows for local storage it does not allow for storage of trees on a server.

Finally, Attack Tree Designer was developed as a module within the Modelio modelling environment. It facilitates the creation and manipulation of ADTs and ”contains additional set of features that allow users to configure security attributes for the attacks such as the severity and the likelihood of the attack.” [CB]. There is a feature that allows for importing and exporting to and from the local file system but no such feature exists for server storage.

3.3 XML

Nghi Tang Le wrote a master thesis on XML resource management. In this paper the aim was to ”analyze the existing technologies and determine how they can be leveraged to solve the problems associated with managing large collections of XML files.” [SS23]. They came to the conclusion that a hybrid database, where XML documents can coexist with relational resources within the same database, was the solution for their web application. However, we have a different set of requirements for both our web application and our storage. E.g. we require only the entire XML as a whole and there is no need for modifications of existing XML files.

Bourret [Bou99], in his older paper on XML and databases, explains that there are three ways to store XML documents. They can be stored in the file system, when you do not have an enormous quantity of documents and have no need for features such as multiple user access, transactions and queries. Another option is to store them in a relational database if ”you are interested in the data, not how it is stored in an XML document.” He advises to use a native XML database, which could be motivated by having XML documents that are difficult to map to a relational structure. As specified in section 5.2 this solution was considered but not selected for various reasons described there.

There is a gap in the research regarding ADT-usage in the form of a missing collection of ADTs. Such a collection would enable to advance the research by facilitating analysis on user-generated ADTs. To the best of our knowledge there are no tools that provide centralised storage in which such a collection could be realised. This is a practical gap. This thesis addresses the research gap by investigating the requirements of a centralised storage solution. The practical gap is then filled by using the requirements in the development of a possible implementation of the storage.

4 Theory

There are several options regarding the storage of XML documents. In this section we will cover them in order to present their advantages and disadvantages. Additionally we will present what we would need to do in case of implementation.

4.1 File system

A possible solution would be to simply store the XML files in a directory on the file system on the LIACS server. As long as the amount of documents is not too great this is the simplest solution[Bou99]. If chosen as solution to be implemented, we would need the ADT Web App to communicate with the server. We would need to make sure that all the actors specified in the Use Case Diagram (see Figure 3) have access to the location of the files with both read and write rights.
Some advantages of this option:

- Storage in the file system is simple.
- Ample storage is available; there is no limit on tree size (excepting the limit of the entire file system).
- There is no need for other software.

Some disadvantages of this option:

- A file system has limited scalability; once the quantity of documents goes up the file system has its limits regarding for example searching [SS23][Bou99].
- The LIACS server is not accessible from outside without the proper credentials.

4.2 Relational database (stuffing)

For larger quantities of data usually a relational database is used. An example of a type of relation database is MySQL, which is available at LIACS. A relation database is ideal for storing data that have relationships in them. For example, storing all the orders for a company. An order has a relationship with the client and with the products.
However, our XML files contain tree structures. These are human-generated documents which are unstructured in the sense that they are difficult to map to different tables and the relationships between them. Note that it is technically feasible to use a relational database like MySQL without defining relations.
According to Buljic and Filipovic there are two options for storing an XML document in a traditional relational database; stuffing and shredding the file[BF12]. This distinction was also made by Nghi Tang Le in their thesis discussing the ideal storage solution for XML generated by a web application[SS23]. We will present both options here as well, starting with stuffing.

Stuffing is known as ”storing XML in a large objects such as TEXT or Large Objects (LOBs) columns”[BF12] which, as the name implies, entails the stuffing of the entire XML file into a large object. For us this could mean to use the MySQL database at the LIACS where we would only

need one table for all the trees. We could use the MySQL- equivalent of a TEXT object to store our XML in a text format.
Some advantages of this option:

- Stuffing works well ”when entire document is required for processing” [BF12].
- It is relatively simple to ”stuff”; it is analogue to storing the XML in the file system but with some added advantages.
- MySQL is already available at the LIACS so there is no need for installing other instances of a relational database.

Some disadvantages of this option:

- A relational database is traditionally not meant for unstructured data.
- The LIACS server is not accessible from outside without the proper credentials.
- One cannot search within the XML itself.

4.3 Relational database (shredding)

The counterpart to stuffing is called shredding. In an attempt to structure the XML so that it can be mapped to tables and relations ”one must first shred an XML tree-structured document so that it fits into flat relational tables.” [FS02]. The file is shredded into multiple parts which are subsequently mapped onto the tables and columns in the database [BF12]. For a complicated XML file the number of tables, columns and relationships can get very large. This in turn makes searching for information more complex since the SQL statements required will be also increase in complexity [BF12][SS23].
In our case we could still make use of the MySQL database currently present at LIACS. However, we would need to write a parser that takes care of translating the XML generated by the ADT Web App into a relational structure. Here lies a complication; since our data is unstructured by nature, as discussed before, it would be complex to build this parser.
Some advantages:

- Searching for information within the XML can be fast once one has the necessary queries [BF12].

Some disadvantages of this option:

- Shredding is not suitable for unstructured data.
- Building the parser required for shredding would be complex.
- Reconstructing the XML document for retrieval might take some time due to the complex SQL statements required.

4.4 Native XML database

These databases are designed to handle XML and the querying of the data. They are fundamentally different from relational databases in the sense that it ”has an XML document (or its rooted part) as its fundamental unit of (logical) storage and defines a (logical) model for an XML document, as opposed to the data in that document (its contents).” [PL07]. There are quite a number of them available, for example Oracle [SS23] has one.

Selecting this option would imply that we make some configuration considerations. Some are commercial services that need to be purchased but some others are open source, like eXist (https://exist-db.org/exist/apps/homepage/index.html). Once we have selected the preferred one we need to install it on the server of LIACS and configure it to our needs. It then needs to be able to communicate with the ADT Web App to facilitate the storing of the trees.

Some advantages of this option:

- A native XML database has good scalability; they ”are also highly scalable and can handle very large volumes of data efficiently” [SS23].
- Some native XML databases are open-source and freely available.

Some disadvantages of this option:

- LIACS does not have a native XML database installed.
- Native XML databases of more well-known companies like Oracle need to be purchased.
- The learning curve for using a native XML database might be steep.

4.5 Hybrid of native XML and relational database

There are some relational databases that are ”extended by native XML data type for storing and manipulating of XML documents.” [BF12]. This kind of database is highly suitable for applications that use both XML and other data [PZP09]. Papamarkos et al. describe a couple of these hybrid databases from notable players in the field like Oracle and IBM [PZP09].

If chosen, this option will approximately follow the same path as the native XML database option. We would need to choose the one that best fits our requirements and needs. Once the choice is made it would need to be installed and configured on the LIACS server. Afterwards the communication with the ADT Web App would need to be established.

Some advantages of this option:

- A hybrid database is suitable for applications that work with both other kinds of data and XML.
- They are available from large companies which means there is much support.
- A hybrid database has good scalability.

Some disadvantages of this option:

- Hybrid XML databases of more well-known companies like Oracle need to be purchased.

In summary, there are five options that we can choose from. We listed them below in Table 1. After we have specified the requirements in section 5.1, we will consider all the options in relation to the requirements to see which one best fits our needs.

Option	**Keywords**
1	File system
2	Relational database - stuffing
3	Relational database - shredding
4	Native XML database
5	Hybrid database

Table 1: Summary of the possible solutions

5 Methodology

This thesis progressed through four stages. The project started with requirements elicitation from the stakeholders. Once the requirements were known research was conducted to identify a viable solution for the storage of the ADTs as XML files. When the theoretical solution met all the requirements and was accepted by our supervisor the project moved into the prototyping phase. The prototype was build locally and was meant to reveal any potential obstacles that would halt development early on. Once the prototype proved the solution to be viable and was to the satisfaction of our supervisor we could start implementing the final deliverable. This was then integrated into the ADT Web App.

We believe that the process of turning an idea into a deliverable is not a straight line. In our experience sometimes one has to go back to a previous stage or one can be in two stages simultaneously. However, for the sake of the readability of this thesis we will present the stages in linear fashion.

5.1 Phase 1; requirements elicitation

As specified above, the initial phase of the project entailed eliciting all the necessary requirements for the storage solution.
Given the many ways to implement a feature/product there is a need for a framework of requirements that works like a search filter in order to limit the amount of possible solutions. Using the RE process described in section 2 we captured the requirements from our thesis supervisor. Only the possible solution that adhere to the requirements are worth looking into; everything else would be futile. If the requirements are not specific enough the project will lose much of its viability since time and energy will be spent on unnecessary aspects.

For the storage solution for the ADT Web App there were several meetings to discuss the requirements and their motivation. Because the requirements directly influenced the design choices of the implementation we will discuss the key requirements in order of importance:

1. **Security**
   Our solution must be secure; this is valid for both the data and the system. The data must keep its integrity, no user can compromise the data. System security must also be maintained; no unauthorised user can access the LIACS systems.

2. **Implementation must work with existing resources.**
   Bearing in mind that this thesis is part of an ongoing project we have to ensure that any solution we propose works with the existing resources allocated for this project. For example, solutions that intend to use services which require additional payment, be it once or on a subscription basis, are considered not viable. We assume that these payments are not within the allocated resources. Another example are solutions that require installation and maintenance of special software.

3. **The XML format of the ADTs cannot be changed and there should be no data loss.**

The XML format of the trees must remain compatible with other applications such as ADTool [KKMS13]. Furthermore, we do not want the ADTs to be incomplete in storage.

4. **Simplicity.**
   We want the solution to be as simple as possible; its development must stay within the time constraint of this thesis and the end users must be able to handle the storage with ease.

5. **Users of the ADT Web App do not have access to the storage.**
   This requirement stems from a security standpoint. ADTs may contain sensitive information of a company's cyber defences (or lack thereof). It has to be avoided that company A gets access to this information of company B. The storage is therefore not accessible for users, using what we call the ”tree identifier” (more on that in section 5.3) users can only submit and retrieve their own ADTs.

6. **The ADT Web App will not have user accounts.**
   To keep user-friendliness as high as possible user accounts were not implemented in the ADT Web App; nor will they be. Furthermore, this avoids privacy concerns. This requirement is relevant for implementation because it makes it impossible to determine the owner of a submitted tree. Therefore, the tree identifiers were implemented.

7. **Users must be able to submit their ADT to the storage**.
   Since the ADT Web App also has a feature that allows the users to download and upload their trees to and from local storage we will use different terminology to avoid confusion. We refer to submitting or uploading to the server when the tree goes from the ADT Web App to the storage. We refer to retrieving or downloading when the tree goes from the storage to the ADT Web App.

8. **Users must be able to retrieve their ADT from the storage.**
   Motivation is identical to the previous requirement.

9. **Users are asked for consent before submitting their ADT.**

10. **LIACS' researchers must have full access to the storage.**
    In order for the data set to be used for analysis it must be accessible for analysts. Since the LIACS servers are not publicly accessible, researchers will have to contact the LIACS research team in order to obtain the necessary credentials that will provide them access to the data set.

To summarise the requirements of the storage we made a Use Case Diagram (UCD), which is a certain type of diagram from the Unified Modeling Language (UML) [BJR+96]. To have a visual representation of the requirements helped both in meetings and in the process of implementation. The UCD is displayed in Figure 3.

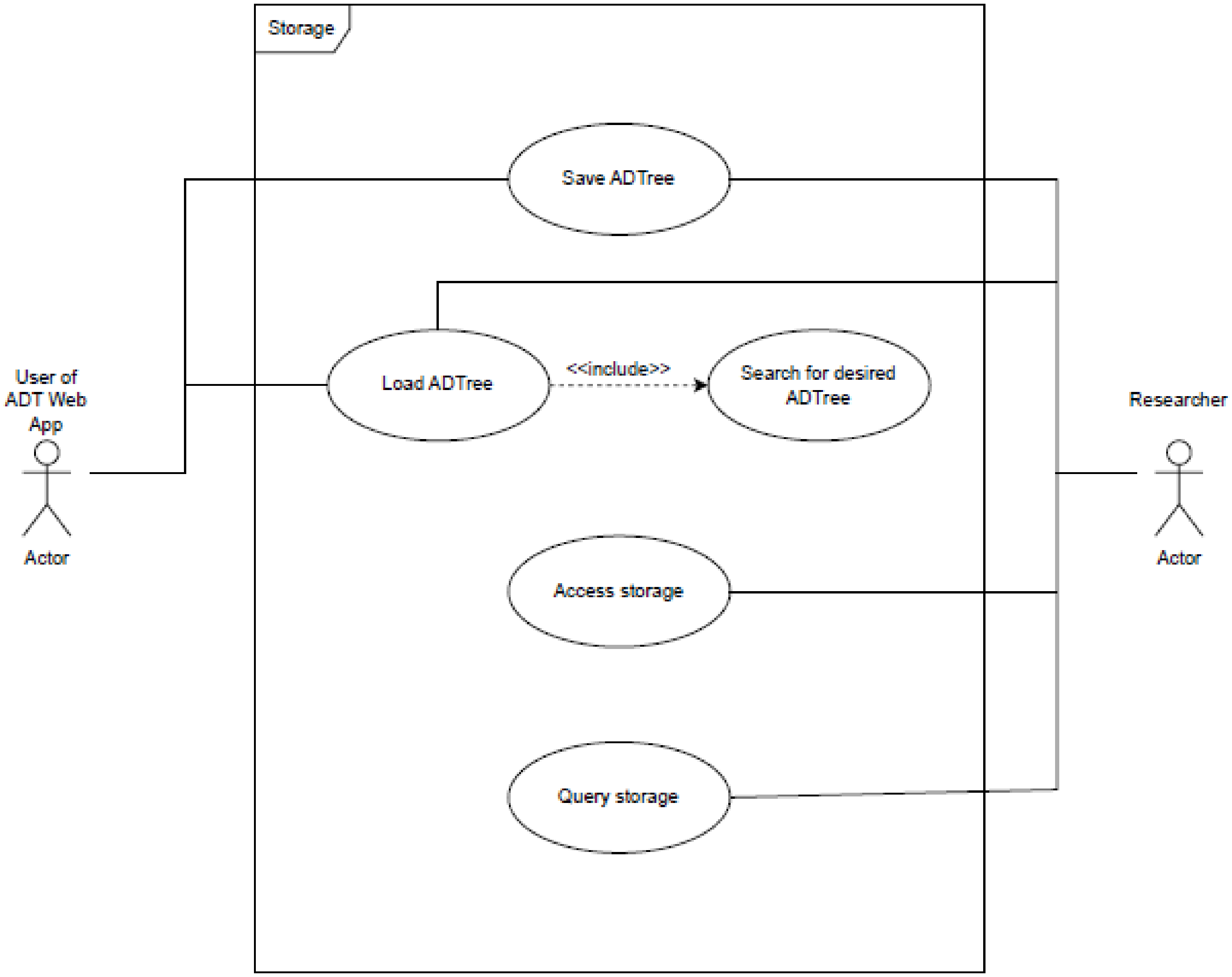


Figure 3: UCD of the requirements

On the left of the figure one of our two actors (users) is shown: the user of the ADT Web App. On the right side shows the other actor: a researcher. The circles represent use cases, which are actions that the actor wants to perform with the system. E.g. the user of the ADT Web App wants to be able to save their ADT; this is displayed with a line extending from the actor to the circle containing "Save ADTree". Between the use cases "Load ADTree" and "Search for desired ADTree" is a so-called include relationship. This entails that in order to achieve the use case at the left side of the arrow the actor must also achieve the right side of the error. In our UCD this signifies that in order to load their ADTs an actor must also be able to specify which one is theirs.

5.2 Phase 2; find a solution

After the requirements have been determined the next phase is to find a viable solution to the problem. This iterative phase contains brainstorming, researching, sparring with team members and (in non end-to-end approaches) many failed attempts. During this phase contact with the stakeholders may be minimal since they have no need to be involved with every attempt at a solution. Once a potential viable solution (or a selection for comparison) is found, it will be discussed with the stakeholders which might lead to revisions or back to the drawing table [INC23].

In following parts we will discuss our path to the eventual solution that was implemented. We will cover all the potential solutions that were considered with their trade-offs. Finally, we will present our solution that made it into the next phase.

5.2.1 Trade-off

Table 2 contains the trade-off between the options described in section 4 and the requirements listed above in section 5.1. For every satisfied requirement the option is awarded with a plus (+) and it gets a minus (-) if the requirement is not satisfied. As stated, the requirements are in order of importance.

Option	**Keywords**	**Requirement**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	File system	-	+	+	+	+	+	+	+	+	+
2	Relational database - stuffing	+	+	+	+	+	+	+	+	+	+
3	Relational database - shredding	+	+	-	-	+	+	+	+	+	+
4	Native XML database	+	-	+	-	+	+	+	+	+	+
5	Hybrid database	+	-	+	-	+	+	+	+	+	+

Table 2: List of requirements satisfied (+) or not satisfied (-) per option

For requirements 5-10 the options are not distinctive; they are satisfied in all the options. We will therefore discuss only the requirements that made a difference.

5.2.2 Security

If the security of either the data itself or the system is compromised, then no other requirements matter. Thus option 1 (file system) was rejected immediately due to the violation of requirement 1 (security).
Additionally, it is not desirable for the ADT Web App to directly access the database on the server since all code and documentation is publicly available on Github. This would violate requirement 1 (security) specified in section 5.1. To circumvent this we decided to let the ADT Web App communicate with a PHP script located on the LIACS web server. The PHP script in turn would take care of the database transactions.

5.2.3 Implementation must work with existing resources

Since LIACS does not have a native XML database or a hybrid database installed, we would need to install one. We would also need to configure and maintain it. This is a violation of requirement 2 (existing resources) as specified in section 5.1.

5.2.4 The XML format of the ADTs cannot be changed and there should be no data loss

For both the retrieval of trees from storage and for further analysis of the dataset it is important that there is no data loss. As discussed in section 4, shredding consists of breaking the XML file into multiple parts before these parts are mapped onto relational tables and columns. We cannot guarantee that in the process of making these parts all data integrity will remain intact. This would be a violation of requirement 3 (XML format) stated in section 5.1.

5.2.5 Simplicity

The last distinctive requirement was that the solution had to be as simple as possible. As stated before, the file system would be the simplest option. However, option 1 (file system) had already violated requirement 1 (security) and was therefore rejected. Option 3 (shredding) involved building a parser to parse our data to map in onto relational tables. The unstructured nature of the data would make this difficult which violates requirement 4 (simplicity). Option 4 (native XML database) and 5 (hybrid database) were both already rejected due to violation of requirement 2 (existing resources) but since they require installation, maintenance and a learning curve they are also rejected on account of violating this requirement 4 (simplicity).

The trade-off converged to one option that satisfied all the requirements; a relational database where we will use stuffing to store the XML into a single table. The next section 5.3 will discuss in more detail the implementation of the solution.
The solution was now ready to move to the next phase; implementation. Until the implementation was ready it was decided that both the PHP script and the database would first be located on the work area allocated for the student executing this thesis as a prototype. This approach allowed for implementation in a safe environment and optimisation of the solution before moving it to its final location.

5.3 Phase 3; prototype solution

The next step in the process is to start building the implementation. There are many methodologies for doing so, but the one we found to be optimal is end-to-end implementation using the so-called trace bullet approach [TH20]. This approach entails first ensuring to connect all the ends and have them operational before enhancing features on either end as described in Figure 4.
A well-known example of this approach could be a so-called ”Hello World” program (a program which does nothing besides printing ”Hello World” to the screen or terminal). When developing this program one has achieved much more than printing only this simple text. It also demonstrates the successful setup of the project environment, compiler (for programming languages such as C++), and other dependencies such as packages or libraries.

However, in its current state the "Hello World" program does not accomplish anything "real" yet. It does e.g. not handle any data directed towards meeting the users needs [tra]. Figure 4 4 shows an example of the tracer bullet for some generic application [TH20]. The more traditional, alternative approach is to perform more (coding) work upfront followed by the execution of the program as a whole. This may work, like "hoping for the best", but the risk is that the effort produced a product that deviates considerably from the intended implementation. Which in turn potentially leads to negative effects such as frustration of the developer(s), wasted resources and stakeholders that are not satisfied [tra].

Using this approach allows identifies issues that might halt development early on and resolve them before they can become an unsolvable problem. For example; the developers start implementing on one end, making sure that side is completely finished with all features working properly. They then try to connect this to the other end which does not work for whatever reason. In such cases, adjustment on the first end is needed, which in a worst case scenario needs to be redone completely. However, had there been made an attempt to connect them sooner the developers might have found the problem sooner enabling them to build the first end correctly on the first attempt.

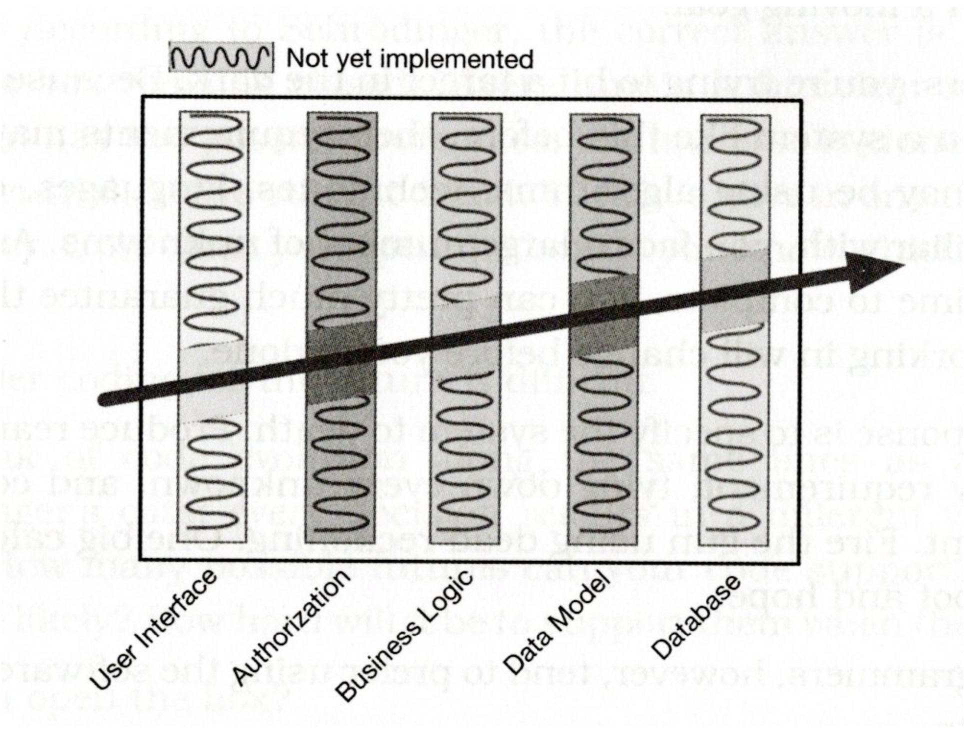


Figure 4: An example of the tracer bullet approach, taken from [TH20]

For our storage solution we took the end-to-end approach. In the previous phase it was decided that we will have the ADT Web App communicate with a PHP script which in turn will handle the insertion or selection of the ADTs. The trees will be sent and retrieved in XML format since the ADT Web App already contained code that renders the XML of the ADT on display and displays a tree when given the XML.

Figure 5 shows a visual representation of the data flow between the three components (ADT Web App, PHP script and the MySQL database).

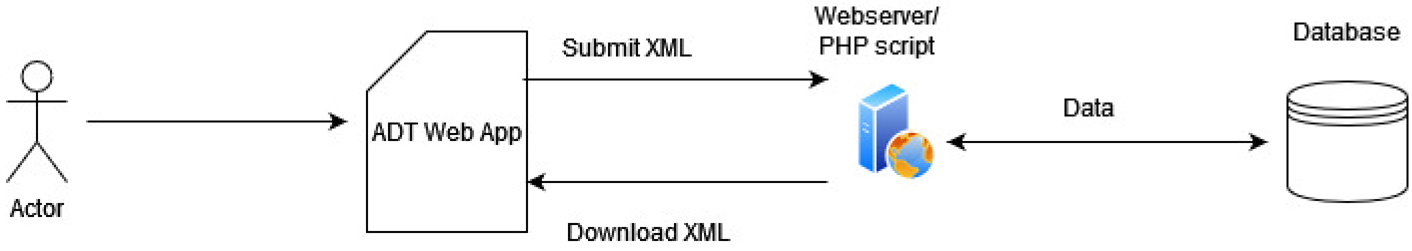


Figure 5: Visual representation of the data flow

5.3.1 Tree identification

Since users cannot access the storage (requirement 5 in section 5.1) and the ADT Web App (requirement 5 in section 5.1) does not support user accounts (requirement 6), we designed a way to implement the retrieval of trees. Once they are submitted it is impossible to retrieve its originator. To this end we created the "tree identifier". When a user submits their tree to the storage they are asked to provide a name for their tree. Subsequently, they are provided with what we refer to as a "token", a random number between 1 (inclusive) and 99999 (inclusive). The tree name and token combined form the tree identifier.
At retrieval, the user provides the (correct) tree identifier and their tree will be displayed in the ADT Web App. There is a minuscule chance that the tree identifier is not unique. When the tree is submitted this does not pose a problem; every insertion generates a new entry due to an auto-incremented column in the MySQL table. Consequently, the database will generate a unique number for every record the moment it is created. This number is also stored in a column (see also table 3) mitigating the problem of an identical tree identifier.
At retrieval it might be a bigger problem, in the event when a user has a tree identifier that coincides with one from a tree from another user. In the extremely rare case this happens the database would return the first tree that matches the tree identifier. This means it could be the correct one or one that does not belong to the user who initiated the retrieval.
We decided not to act upon this issue because the risk is negligible. The probability that the token is identical is 1 in 100000. Furthermore, if this happens then the tree name (which the user can fill in freely) must also be the same. Mitigating this risk would necessitate to check the entire storage before inserting a new tree. This would have led to recycled code and increased communication between the PHP script and the ADT Web App. We found the likelihood of this scenario simply to small to spend more time on. Also this is not a requirement; implementing this would be an example of project creep.

5.3.2 PHP script

We started implementing with the PHP script because that is the central link to connect both the ends. Since its link to the database is crucial for our solution to work we started there. We began by establishing a connection to the database and inserting some test data (normal text) into a test table. This meant we had now implemented (albeit in a rudimentary form) the ”Data” arrow in Figure 5 going from left to right. Keeping in mind the end-to-end philosophy, we then tried to retrieve this test data from the database. When this succeeded we completed both directions of the ”Data” arrow in Figure 5.

5.3.3 XML storage in the database

The left end of the data flow (the ADT Web App) is a fixed component within this implementation so we decided to continue on the right end, the database. This way we could still explore alternative options with minimal energy wasted if this idea did not work out. The primary question here was; how to save the XML in the database? As discussed previously in section 5.2, a tree structure is typically unsuitable for a relational database. However, we do not have to search and/or edit within the trees. Therefore we decided to save the XML as a whole. MySQL lacks a XML datatype so we opted for the VARCHAR datatype. In contrast to the CHAR datatype, VARCHAR does not use padding on the data if the data require less storage than declared[mys]. We preferred this option because the length of the XML can vary greatly.
Using the PHP script we extracted the XML from a test file placed on the web server and inserted it into the database. The XML was inserted successfully and it seemed that the format was preserved without any discernible alterations: it looked the same in the test file as in the database. Keeping the end-to-end approach in mind, we decided that to verify the functionality we would start building the retrieve function in the ADT Web App (the ”Download XML” arrow in Figure 5). As a result, we could then ascertain whether we could retrieve and correctly display the inserted test tree. Listing 1 illustrates the representation of the tree from Figure 2 as it is stored in the database.

Listing 1: The XML of an ADT in the database

```
<?xml version=”1.0”?>
<adtree>
  <node refinement=”disjunctive”>
    <label>Bank Account</label>
      <node refinement=”conjunctive”>
        <label>ATM</label>
          <node refinement=”disjunctive”>
            <label>Card</label>
          </node>
          <node refinement=”disjunctive”>
            <label>PIN</label>
              <node refinement=”disjunctive”>
                <label>Find Note</label>
                  <node refinement=”disjunctive” switchRole=”yes”>
                    <label>Memorize</label>
                  </node>
              </node>
              <node refinement=”disjunctive”>
                <label>Force</label>
              </node>
          </node>
      </node>
      <node refinement=”conjunctive”>
        <label>Online</label>
          <node refinement=”disjunctive”>
            <label>Username</label>
          </node>
          <node refinement=”disjunctive”>
            <label>Password</label>
              <node refinement=”disjunctive”>
                <label>Phishing</label>
              </node>
              <node refinement=”disjunctive”>
                <label>Key Logger</label>
              </node>
          </node>
          <node refinement=”disjunctive” switchRole=”yes”>
            <label>2FA</label>
              <node refinement=”disjunctive” switchRole=”yes”>
                <label>malware</label>
                  <node refinement=”disjunctive”>
                    <label>Mobile</label>
                  </node>
              </node>
          </node>
      </node>
  </node>
</adtree>
```

5.3.4 Retrieving a tree

To make the link between the ADT Web App and the PHP script we made use of JavaScript's ability to execute network requests. Using the *fetch* function we developed code that made a GET request from the ADT Web App to the PHP script hosted on the web server. Subsequently, we programmed the PHP script to return our manually inserted test tree. It is required to set the CORS-header in the PHP script since the ADT Web App and the script are not on the same domain, otherwise the requests are blocked.
Following the creation of a clickable button on the toolbar of the ADT Web App (see Figure 7) that invoked the JavaScript code we succeeded in correctly displaying the tree in the ADT Web App. This marked a first working implementation of the ”Download XML” arrow in Figure 5 which could be expanded to meet the requirements stated in section 5.1. However, as end-to-end protocol dictates, first we developed remaining features before expanding existing features. This meant that

we had to implement the ”Submit XML” arrow of Figure 5.

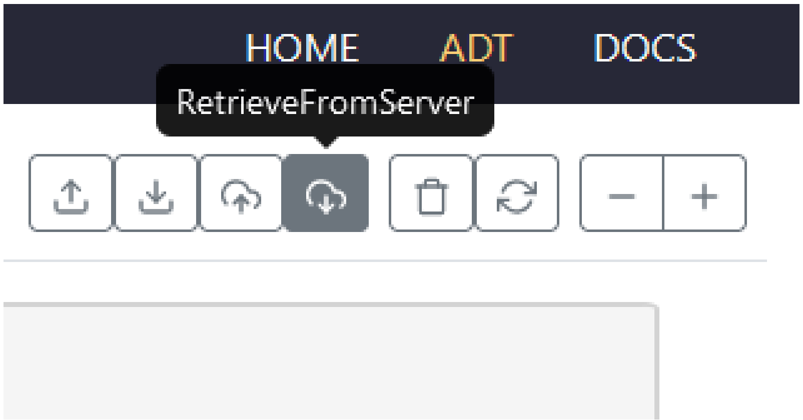


Figure 6: The retrieve button in the ADT Web App

5.3.5 Submitting a tree

To implement this we utilised the *fetch* function again, this time configuring it to make a POST request to the PHP script. For this purpose we created another clickable button on the toolbar of the ADT Web App (see Figure 7). We expanded the POST request step by step; initially sending a short string, then sending the XML of a tree and ultimately the insertion of this XML into the database. Upon the successful completion of these steps we had realised the ”Submit XML” arrow of Figure 5.
At this point all the components from Figure 5 and links between them were operational. Due to the end-to-end approach implementation proceeded relatively smoothly without any issues of major impact. We could start expanding all our components so that they would meet all the requirements. Still adhering to end-to-end, we started at the database end with designing the table that would contain the ADTs.

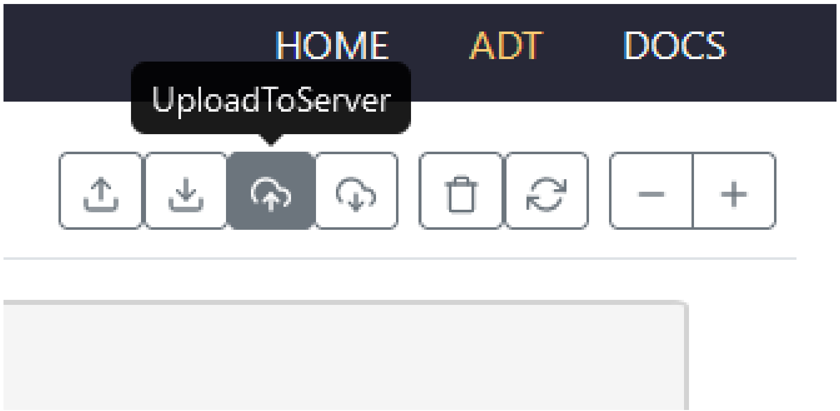


Figure 7: The upload button in the ADT Web App

5.3.6 Database structure

Our table has five columns; an ID, the username, the tree-name, the token and the XML. Additionally, we request the user to provide their name in case they lose their tree identifier. They can contact the storage maintainer who can then search more easily for the required tree. The data types of the columns are BIGINT, CHAR, CHAR, CHAR and VARCHAR respectively. We chose BIGINT for the ID column to accommodate $2^{64} - 1$ unique entries in the database (assuming the table limit, which is regulated by the operating system, has not been reached).
The maximum size of a row in a MySQL database is 65535 bytes, which is $2^{16} - 1$ as specified in the MySQL 8.0 Reference Manual [mys]. We take 128 bytes for the tree identifier and the other household items; 8 bytes for the ID, 32 bytes for the username, 64 bytes for the tree-name and 5 bytes for the token. Consequently, roughly $2^{16} - 2^7$ bytes remain for storage of the tree itself. Table 3 provides a summary of the above. The ADT in Figure 2 has 15 nodes and its corresponding XML has 1571 characters, meaning that 1 node generates roughly 100 characters in the XML. So our storage can store ADTs with a maximum of around 650 nodes.

Column name	**Datatype**	**Size in bytes**
ID	BIGINT	8
Username	CHAR	32
Treename	CHAR	64
Token	CHAR	5
XML	VARCHAR	$2^{16} - 2^7$

Table 3: Summary of the database structure

5.3.7 User Interface

We then proceeded to expand our POST request to include the username, tree-name and token. All of these are generated on the ADT Web App side. We implemented prompts to solicit user consent and to collect their username and tree-name. If the user declines consent, nothing else happens. In cases where the user does give consent but fails to provide a username and/or tree-name, default values are used. All user provided inputs (including the tree) are checked to be within limits.
These ”tree-data” are then attached to the POST request in a JSON format. In turn, the PHP script extracts the data and inserts it into the database using a prepared SQL statement. Upon completion, the user receives a message either stating their ”tree identifier” or an error.

Subsequently, we expanded the functionality of the GET request. We introduced the prompts that ask the user to provide the tree-name and token of their desired tree. The acquired inputs were then incorporated as parameters in the URL of the GET request. The PHP script uses another prepared SQL statement to select the tree that matches the parameters. The ADT Web App verifies whether the response is an error, in which case it relays the error to the user. This can occur for example when the user provides an incorrect tree identifier. Conversely, if the response is acceptable, we verify whether the format of the response is XML. If the format is correct, we display it. Otherwise the user is informed that the format is incorrect and the tree is not displayed.

The implementation had now reached conclusion, marking the end of phase 3 of the project. The entire prototype met the requirements and was to the satisfaction of our supervisor.

5.4 Phase 4; implement deliverable

The last phase of a project consists of implementing the final product. The new product has to be integrated inside the existing process and/or products.
As discussed in section 5.2 the database and PHP script were located on a work area allocated to a specific student. Given that old student accounts are deleted after graduation it was not an option to leave the product where it was. After consulting with our supervisor both the PHP script and the database were relocated to a location on the LIACS server accessible to the research group of which our supervisor is a part.

The code was modified to its final configuration; to let the ADT Web App send the HTTP requests to the new location of the PHP script. A final round of testing was conducted to demonstrate that our feature was finished and ready to be merged into the main branch on GitHub. These tests consisted roughly of the steps taken in section 6.
Integration on Github signaled the end of phase 4. The product had been migrated and continued working as expected. Instructions on how to access the database to extract the trees were provided to the supervisor ensuring that they could do so independently in the future.

6 Result

The result of this thesis is a new feature in the ADT Web App that allows users to submit and retrieve their ADTs from a storage location on the LIACS servers. As a consequence, this thesis contributes to an ever-growing dataset of ADTs which can be used for further research and analysis. Potentially it might even be used as a training set for some model and/or artificial intelligence.

In this section we will showcase our product. Step by step, we will go through the process of submitting a tree, and subsequently, retrieving it from the implemented storage solution.

Figure 8 shows an example of an ADT.

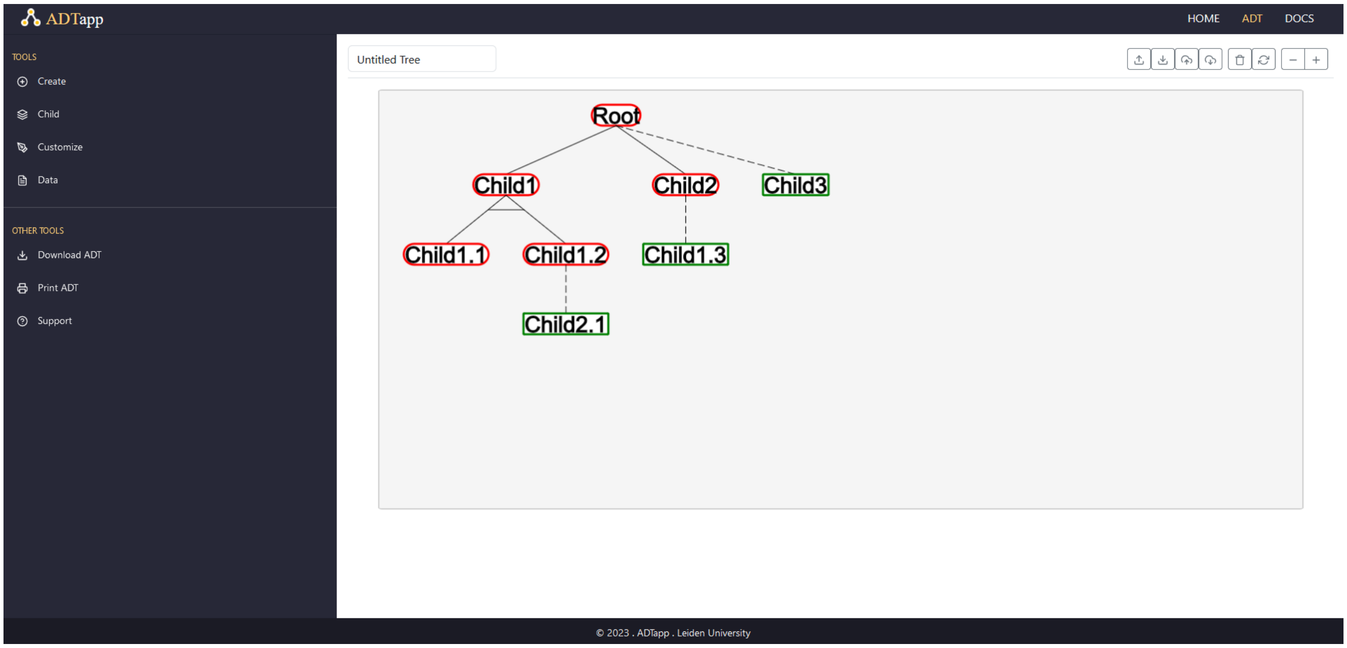
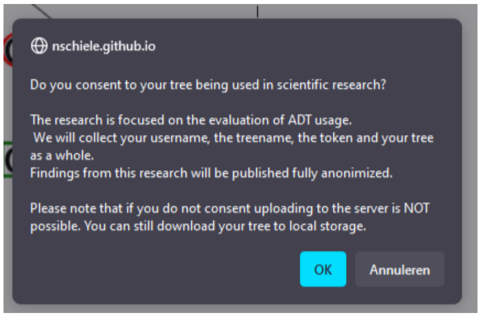


Figure 8: The ADT

The first step in uploading a tree is to press the button located on the upper right corner of the ADT Web App (displayed in Figure 7). This leads to a pop-up containing a consent message, as shown in Figure 9.



nschiele.github.io

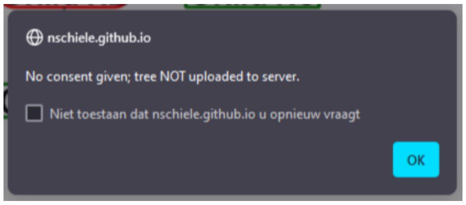
Do you consent to your tree being used in scientific research?

The research is focused on the evaluation of ADT usage.
We will collect your username, the treename, the token and your tree as a whole.
Findings from this research will be published fully anonimized.

Please note that if you do not consent uploading to the server is NOT possible. You can still download your tree to local storage.

Figure 9: The consent message

If the user does not consent, the app presents the user with the following message (Figure 10) and nothing else happens. They can continue to use the ADT Web App and all its features.



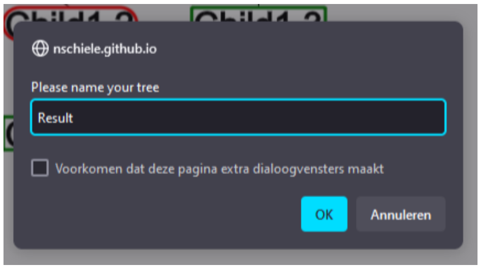
nschiele.github.io

No consent given; tree NOT uploaded to server.

□ Niet toestaan dat nschiele.github.io u opnieuw vraagt

Figure 10: Consent not given

If the user does give consent, the ADT Web App asks to provide a name for the tree. In the example of Figure 11 the tree-name is "Result". In case a name is not filled in the ADT Web App uses the default value "TreeName" to prevent empty values in the database.



nschiele.github.io

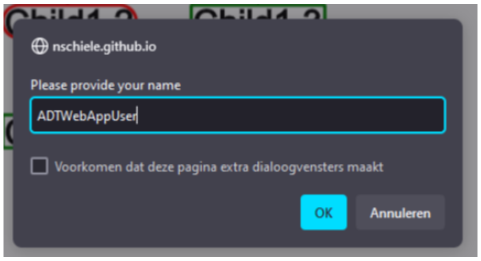
Please name your tree

Result

□ Voorkomen dat deze pagina extra dialoogvensters maakt

Figure 11: Provide a tree-name

Once the tree is named, the ADT Web App asks to provide a username. Excepting the length there are no restrictions or requirements for the username. An example is shown in Figure 12. Again, if the username is not filled in the ADT Web App uses the default value "UserName".



nschiele.github.io

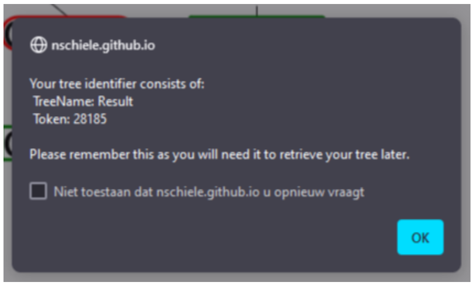
Please provide your name

ADTWebAppUser

□ Voorkomen dat deze pagina extra dialoogvensters maakt

Figure 12: Provide a username

The submission is successful; the user is given their ”tree identifier”. This is shown in Figure 13.



nschiele.github.io

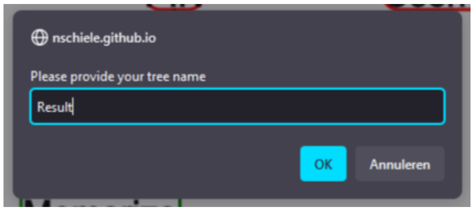
Your tree identifier consists of:
TreeName: Result
Token: 28185

Please remember this as you will need it to retrieve your tree later.

□ Niet toestaan dat nschiele.github.io u opnieuw vraagt

Figure 13: The ”tree identifier”

The process of submitting a tree is now completed. If a user after some time wants to retrieve their ”Result” tree, then the first step is to press the button located on the upper right corner of the ADT Web App (displayed in Figure 6). As Figure 14 shows, the ADT Web App asks for the first part of the ”tree identifier”, which is the tree-name.



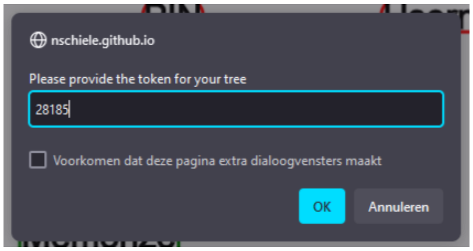
nschiele.github.io

Please provide your tree name

Result

Figure 14: Fill in the tree-name

In the next step (Figure 15), the ADT Web App asks to provide the token, which is the second part of the ”tree identifier”.


nschiele.github.io

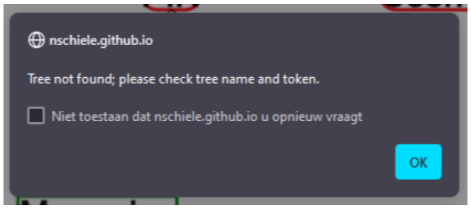
Please provide the token for your tree

28185

□ Voorkomen dat deze pagina extra dialoogvensters maakt

Figure 15: Fill in the token

If the user provided a wrong tree-name and/or token the ADT Web App shows the error displayed in Figure 16.


nschiele.github.io

Tree not found; please check tree name and token.

□ Niet toestaan dat nschiele.github.io u opnieuw vraagt

Figure 16: The tree could not be found

Once the right "tree identifier" is provided the "Result" tree is displayed again in the ADT Web App like in Figure 8.

The above shows the successful implementation of our research goal stated in section 1.1.

7 Discussion

Now that we have shown the result of this thesis we will go over the requirements once more. Per requirement we will specify how we satisfied it and incorporated it into our result.

1. **Security**
   We satisfied this requirement twofold. Firstly we have selected the MySQL database already present at LIACS, because this meets the system security requirements. Secondly by letting the ADT Web App communicate with the PHP script instead of directly with the database we made our solution more secure. This way the sensitive access information (e.g. database password) is not included in the ADT Web App code (publicly accessible on GitHub). The PHP script is not publicly accessible.

2. **Implementation must work with existing resources.**
   We satisfied this requirement by implementing our storage using the MySQL database already available at LIACS. Nothing new needed to be acquired to implement our solution.

3. **The XML format of the ADTs cannot be changed and there should be no data loss.**
   We have not changed the format of the XML at any time during development which adheres to this requirement. By choosing option 2 (stuffing) from the list in section 4 we make sure that there is no data loss. We stuff the XML as a whole into one string. This string is unchanged and only written into the database. Upon retrieval we take the entire string without changing it. Using this method we preserve the XML format.

4. **Simplicity.**
   Simplicity was achieved by selecting the solution that uses stuffing in a relational database. Stuffing is simpler than shredding [BF12]. Furthermore, LIACS already had a MySQL database installed so using that was simpler than selecting, installing and configuring another one.

5. **Users of the ADT Web App do not have access to the storage.**
   By letting the PHP script handle the database in terms of connecting and transactions without user interference we satisfy this requirement. The PHP script is hosted on the LIACS server as well so users have no access to the script either.

6. **The ADT Web App will not have user accounts.**
   With the implementation of the tree identifiers (see Figure 13) described in section 5.2 we circumvented a need for user accounts. Therefore, the implemented solution satisfies this requirement.

7. **Users must be able to submit their ADT to the storage**.
   With the implementation of the feature behind the upload button in Figure 7 we have satisfied this requirement. The tree goes via the PHP script into the database.

8. **Users must be able to retrieve their ADT from the storage.**
   With the implementation of the feature behind the retrieve button in Figure 6 we have satisfied this requirement. The PHP script retrieves the correct tree from the database after which it is displayed by the JavaScript code.

9. **Users are asked for consent before submitting their ADT.** Figure 9 displays the consent message presented to the user. By ensuring that the tree is only stored when consent is given we satisfy this requirement.

10. **LIACS' researchers must have full access to the storage.**
    The instructions and credentials for accessing the database were provided to the research group. If any researchers outside the research group want to access the storage they must contact the research group to obtain the necessary credentials. With that they will have full access to the storage which satisfies the requirement.

The objective of this thesis was to investigate the requirements of a storage solution in order to advance research into usage of ADTs. Our research question was defined as ”how to best implement a storage solution which facilitates future analysis of ADTs generated by the ADT Web App?” (see section 1.1). As shown above and in section 5.1, we have elicited requirements for the centralised storage solution for ADTs. Next, we have used these requirements to develop a possible storage implementation that satisfies the requirements.

The result is an extension to the ADT Web App that generates the collection of ADTs necessary for further research into improving the usability of the ADT.

8 Conclusion and Further Research

This thesis aimed to explain the implementation of a storage solution which facilitates further analysis of ADTs generated by the ADT Web App on two levels using the state of art frameworks in SE and RE. We have followed the higher level of the process that starts with an idea and ends (ideally) with a product and satisfied client. Taking SE principles used on this high level into account we applied them on a lower level. On this low level we explained the specific process of designing, implementing and delivering the storage solution for the ADT Web App.

To achieve this we made use of SE guidelines which translated into four stages of the process. Firstly we elicited the requirements from the supervisor. This was done in meetings with the supervisor conforming to activities described in the INCOSE Handbook[INC23] such as defining stakeholder needs. We went on to find the solution that fit the requirements best within the possible time frame. Following the selection of the solution we designed the implementation of the solution and built a prototype ensuring that we could work safely without disturbing others. The last phase consisted of migrating the product to its final location, transforming it into the deliverable presented in the section 6.

The implementation, following the tracer bullet approach, consisted of building a storage solution within LIACS in the form of a MySQL database. There, the XML of ADTs is stored and accessible to researchers from LIACS. The submitting and retrieving to/from this database is done via a PHP script located on the web server. On the other side of the script is the ADT Web App, which sends POST and GET request with the necessary data. By creating this new feature in the ADT Web App we satisfied the goal of the thesis.

A limitation that our research encountered was due to requirement 3 (XML format), specified in section 5.1. Since the XML format of the ADTs could not be altered we could not keep track of the position of the nodes on the screen. Therefore when a user retrieves their tree from the server all its nodes will be back in the default position.
Another limitation was borne out of requirement 6 (no user accounts). The lack of accounts for the ADT Web App resulted in users having to memorise their tree identifiers. When the number of trees grows it can become a nuisance to have to remember them all. However, there was no way to mitigate this without violating requirement 6 (no user accounts) or requirement 5 (users no access to storage). By giving users access to the storage they could simply look for their trees but this would pose a grave security risk which violates requirements 5 (users no access to storage) and 1 (security). User accounts could have been a way to mitigate the violation of requirement 5 (users no access to storage). Still, that would have violated requirement 6 (no user accounts) stating that the ADT Web App will not have user accounts.

As described before, the ADT Web App is still in development and keeps improving with new features. A suggestion for future work could be to conduct a user study to properly evaluate the usability of the ADT Web App. This could be done by creating a sample containing not only ”technical people” such as students of the Security course but also ”nontechnical people” from various backgrounds such as for example students from different faculties or people selected from certain companies. These people could then use the ADT Web App for a certain amount of time, which could be achieved by giving them an assignment on ADTs. After this time they could be sent a questionnaire asking them about the experience, the usability and whether they have suggestions for improvement. Through this user study we could gain insight into which areas the ADT Web App can still be improved and whether users want more features.

References

[ABP+20] Jaime Arias, Carlos E Budde, Wojciech Penczek, Laure Petrucci, Teofil Sidoruk, and Mariëlle Stoelinga. Hackers vs. security: attack-defence trees as asynchronous multi-agent systems. In *International Conference on Formal Engineering Methods*, pages 3–19. Springer, 2020.

[AYL] D. Leveille A. Yathuvaran, A. Menon and E. Leung. At-at - user manual. https://github.com/yathuvaran/AT-AT/blob/main/Documentation/User

[BF12] Aleksandar Bulajic and Nenad Filipovic. Implementation of the tree structure in the xml and relational database. In *I n SITE 2012: Informing Science+ IT Education Conference*, volume 12, pages 027–051, 2012.

[BJR+96] Grady Booch, Ivar Jacobson, James Rumbaugh, et al. The unified modeling language. *Unix Review*, 14(13):5, 1996.

[BLN+23] Jeremy Bryans, Lin Shen Liew, Hoang Nga Nguyen, Giedre Sabaliauskaite, and Siraj Ahmed Shaikh. Formal template-based generation of attack–defence trees for automated security analysis. *Information*, 14(9), 2023.

[Bou99] Ronald Bourret. Xml and databases. 1999.

[CB] Kaïs Chaabouni and Alessandra Bagnato. Attack tree designer. https://github.com/cpswarm/modelio-attack-tree-module.

[Cen] Nationaal Cyber Security Centrum. Cybersecuritybeeld nederland 2023. https://www.rijksoverheid.nl/documenten/rapporten/2023/07/03/tk-bijlage-cybersecuritybeeld-nederland-2023.

[DZ13] Suguo Du and Haojin Zhu. *Security Assessment via Attack Tree Model*, pages 9–16. Springer New York, New York, NY, 2013.

[FFD+22] Delsina Faiza, Geovanne Farell, Vera Irma Delianti, Sandi Rahmadika, et al. Eight golden rules interface analysis for video conference information system. In *2022 International Conference on Electrical Engineering and Informatics (ICELTICs)*, pages 131–135. IEEE, 2022.

[For] World Economic Forum. 7 trends that could shape the future of cybersecurity in 2030. https://www.weforum.org/agenda/2023/03/trends-for-future-of-cybersecurity/.

[FS02] Juliana Freire and Jérôme Siméon. Adaptive xml shredding: Architecture, implementation, and challenges. In *Workshop on Data Integration over the Web*, pages 104–116. Springer, 2002.

[fST] National Institute for Standards and Technology. Guide for conducting risk assessments. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-30r1.pdf.

[GQZ18] Chao Gong, Yue Qiu, and Bin Zhao. Establishment of design strategies and design models of human computer interaction interface based on user experience. In *Design, User Experience, and Usability: Theory and Practice: 7th International Conference, DUXU 2018, Held as Part of HCI International 2018, Las Vegas, NV, USA, July 15-20, 2018, Proceedings, Part I 7*, pages 60–76. Springer, 2018.

[INC23] *INCOSE Systems Engineering Handbook*. John Wiley and Sons Ltd, 2023.

[iso] Information security, cybersecurity and privacy protection — guidance on managing information security risks. https://www.iso.org/standard/80585.html.

[KKMS13] Barbara Kordy, Piotr Kordy, Sjouke Mauw, and Patrick Schweitzer. Adtool: Security analysis with attack-defense trees (extended version). *arXiv preprint arXiv:1305.6829*, 2013.

[KKvdB16] Barbara Kordy, Piotr Kordy, and Yoann van den Boom. Sptool–equivalence checker for attack trees. In *International Conference on Risks and Security of Internet and Systems*, pages 105–113. Springer, 2016.

[Kli91] George J Klir. *Facets of systems science*. Plenum Press, 1991.

[KMRS11] Barbara Kordy, Sjouke Mauw, Saša Radomirović, and Patrick Schweitzer. Foundations of attack–defense trees. In *Formal Aspects of Security and Trust: 7th International Workshop, FAST 2010, Pisa, Italy, September 16-17, 2010. Revised Selected Papers 7*, pages 80–95. Springer, 2011.

[KS12] Piotr Kordy and Patrick Schweitzer. The adtool manual. *University of Luxembourg*, 2012.

[Mad] S. Madnick. Why data breaches spiked in 2023. https://hbr.org/2024/02/why-data-breaches-spiked-in-2023.

[MO06] Sjouke Mauw and Martijn Oostdijk. Foundations of attack trees. In *Information Security and Cryptology-ICISC 2005: 8th International Conference, Seoul, Korea, December 1-2, 2005, Revised Selected Papers 8*, pages 186–198. Springer, 2006.

[Moh] Marwa Mohalaia. Implementing a user interface for attack-defense tree webapp using human-computer interaction principles. Bachelor's thesis. Bachelor thesis by student at LIACS (2023).

[Mor] Steve Morgan. Cybercrime to cost the world 10.5 trillion annually by 2025. https://cybersecurityventures.com/cybercrime-damage-costs-10-trillion-by-2025/.

[mys] Mysql 8.0 reference manual. https://dev.mysql.com/doc/refman/8.0/en/column-count-limit.html.

[Ore] Stephan Gabriel Meza Orellana. Adtlang: A declarative language to describe attack defense trees. Bachelor's thesis. Bachelor thesis by student at LIACS (2023).

[PAV15] Sophie Pinchinat, Mathieu Acher, and Didier Vojtisek. Atsyra: An integrated environment for synthesizing attack trees: (tool paper). In *International Workshop on Graphical Models for Security*, pages 97–101. Springer, 2015.

[PL07] Gordana Pavlovic-Lazetic. Native xml databases vs. relational databases in dealing with xml documents. *Kragujevac Journal of Mathematics*, 30:181–199, 2007.

[PZP09] George Papamarkos, Lucas Zamboulis, and Alexandra Poulovassilis. Xml databases. *London, UK: School of Computer Science and Information Systems*, 2009.

[Sch99] Bruce Schneier. Attack trees. *Dr. Dobb's Journal*, 1999.

[SS23] Kari Systä and Niko Siltala. Hybrid database for xml re-source management. 2023.

[Sta23] I Standard. Systems and software engineering–system life cycle processes. *ISO Standard*, 15288, 2023. https://www.iso.org/standard/81702.html.

[SW87] Ben. Shneiderman and Carol. Wald. *Designing the user interface : strategies for effective human-computer interaction.* Addison-Wesley, 1987.

[TH20] Dave Thomas and Andy Hunt. *The Pragmatic Programmer.* Addison-Wesley, 2020.

[tra] The power of tracer bullets in pragmatic software engineering. https://medium.com/@remind.stephen.to.do.sth/targeting-success-the-power-of-tracer-bullets-in-pragmatic-software-engineering-cd6c53758986.

[whi] National cybersecurity strategy 2023. https://www.whitehouse.gov/briefing-room/statements-releases/2023/03/02/fact-sheet-biden-harris-administration-announces-national-cybersecurity-strategy/.

[Wym93] A Wayne Wymore. *Model-Based Systems Engineering*, volume 3. CRC Press, 1993.

[You89] Edward Yourdon. *Modern structured analysis.* Yourdon press, 1989.