Opleiding Informatica

# Development of User-Friendly Interfaces In a Technical Tool Used To Create Attack Defense Trees for Non-Technical Users

Reshit Fazlija

Supervisors:
Nathan D. Schiele & Dr. Olga Gadyatskaya

BACHELOR THESIS

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

21/08/2024

## Abstract

In the realm of cybersecurity, the creation and understanding of Attack-Defense trees (ADTs) serves as a critical tool. However, traditional interfaces for designing these trees often prove challenging for non-technical users, hindering their accessibility and usability. This thesis addresses this issue by focusing on the development of a user-friendly web application interface made specifically for non-technical users. Using principles from Human-Computer Interaction (HCI) and leveraging methodologies such as persona-driven design and iterative prototyping, this paper aims to enhance usability and accessibility without compromising the ability to create fully-fledged ADTs, and the ability to expand the tool in the future.

This study evaluates the effectiveness of the newly designed interface by comparing it with interfaces of existing tools, through user evaluations. Key metrics such as usability, user satisfaction, and ease of learning are assessed to demonstrate improvements over existing ADT tools. Results indicate that the new interface significantly enhances user experience, leading to higher usability ratings and increased user satisfaction among both technical and non-technical users.

Using insights gained from the design process and user feedback, this thesis not only presents an improved interface for ADT creation, but also contributes to the broader field of study on interface design for technical tools aimed at diverse user groups. The findings underscore the importance of user-centered design in improving the accessibility of technical tools, particularly in domains as critical as cybersecurity.

# Contents

# 1 Introduction

Attack-Defense Trees (ADTs) are vital tools in cybersecurity, allowing security experts to analyze and mitigate threats. However, existing interfaces for creating ADTs often present usability challenges, particularly for users without technical expertise. This thesis aims to address these issues by developing a user-friendly web application interface made specifically for non-technical users. By applying principles from Human-Computer Interaction (HCI) and utilizing persona-driven design and iterative prototyping, this research seeks to improve accessibility without compromising the functionality of ADTs. Through comparative evaluations with existing tools, this study demonstrates how enhancing interface design can significantly improve user experience and accessibility in the realm of cybersecurity.

The research question addressed in this thesis is: *How can we design intuitive and interactive visualizations for Attack-Defense Trees that make complex cybersecurity concepts accessible to users with varying levels of technical expertise?* This question aims to develop an interface that simplifies the interaction with Attack-Defense Trees, enabling users, regardless of their technical background, to effectively engage with and understand these cybersecurity tools.

The goal of the research is to create an intuitive interactive visualization that allows users of varying technical expertise to use and modify Attack-Defense Trees. The principles, techniques and process of the research will contribute to the field of interface design research, by allowing designers to make their designs intuitive and accessible to many more users.

## 1.1 Thesis structure

To address the research question, the thesis is divided into 8 chapters.

Chapter 1: Introduction - introduces the research topic and presents the research question. It also discusses the significance of the study and provides an overview of the thesis structure.

Chapter 2: Background - explores multiple concepts whose understanding is important to this thesis. It explores Attack-Defense trees, their application and significance in cybersecurity, as well as Human-Computer Interaction principles relevant to interface design and usability.

Chapter 3: Related work - presents a review of existing research and tools related to Attack-Defense Trees and interface design. This chapter highlights gaps in the current literature and how this thesis aims to address them.

Chapter 4: Methodology - presents a general overview of the applied process of the thesis, followed by an expansion on the theoretical framework and an explanation of the evaluation process.

Chapter 5: Implementation - details the full process of design the interface, divided into the three requirement elicitation techniques applied: sketches, mock-ups and personas. Each subsection presents the chronological evolution of the requirements garnered through the use of each technique. In the final subsection, the evolution of the applied technologies, their purpose and role within the projects, and their importance are explained.

Chapter 6: Results - presents the results of the thesis. First, the interface and its individual parts are shown and explained. Second, the interface is compared to previous interfaces, to better highlight the improvements made. Third, the evaluation of the interface is detailed, including both the numerical results, as well as the verbal explanations given during the evaluation. Finally, the limitations of the work are presented, both technical challenges and design challenges.

Chapter 7: Discussion - interprets the findings from the results chapter, discusses the contributions of the research to the field, relates the findings to the initial research question and discusses the validity of the research.

Chapter 8: Conclusions and further research - summarizes the key findings of the study, and suggests areas for future research.

# 2 Background

## 2.1 Attack-Defense trees

Attack-Defense Trees (*ADTrees* or *ADTs*) are an expansion upon the more well known structure of Attack trees. Attack trees are a formal and methodical way of modelling and evaluating the different ways in which a system can be attacked, introduced by B. Shneier [18, 19]. The structured form of these trees is very useful in practice when modeling security scenarios. It can also be an appealing step towards automating analysis of such scenarios [16].

ADTs expand on these attack trees by allowing the addition of nodes that represent defensive measures. This addition enlarges the model capabilities of regular Attack-Trees to represent more complex interactions between attackers and defenders [9].

ADTrees are, in graph theory, connected acyclical graphs. That means that all nodes are connected to any other node by exactly one path. In more basic terms, ADTrees follow a basic tree structure, where one node is designated as the 'root' node, and from there the tree branches into multiple other nodes, which themselves can continue to split. Nodes that have no children are normally referred to as leaf nodes, however ADTrees do not differentiate between leaf and non-leaf nodes.

Each node is either an attack node or a defense node. Attack nodes describe offensive steps that an attacker might take to attack a system. An example of such a node might be: "Impersonate a bank teller." Defensive nodes describe steps that the defending party might take to prevent an attack. An example would be: "Use bio-metric authentication to confirm employee identities."

Nodes have two basic features to represent an ADTree: refinements and counter-measures. Refinements describe the relationship between children of a certain node, which are of the same type as the parent node. An attack node with 3 attack node children and an 'AND' refinement describes an attack broadly in the parent node and then contains the steps of the attack within the child attack nodes. The AND refinement is often visualized through the use of a horizontal line connecting the different children. An example of such a structure can be found in Figure 1.

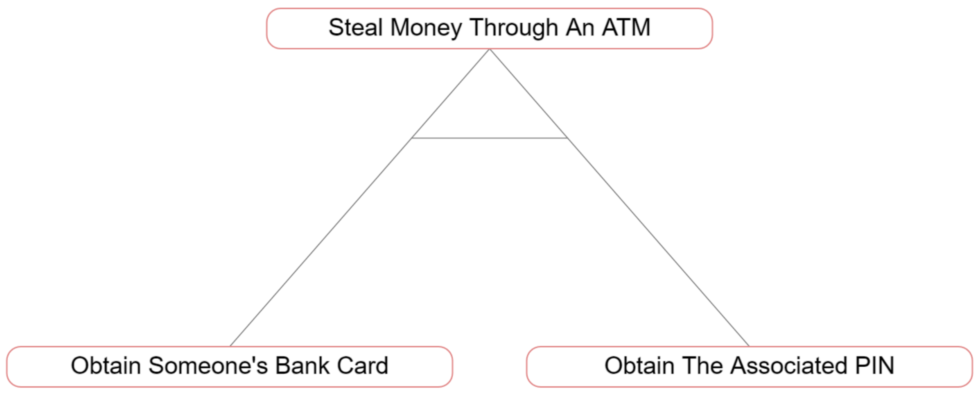

Steal Money Through An ATM
Obtain Someone's Bank Card
Obtain The Associated PIN

Figure 1: An ADTree showing an attack node with the AND-refinement. The children describe necessary steps: stealing someone's money through an ATM, requires an attacker to obtain someone's bank card *and* PIN number.

An attack node with an 'OR' refinement describes an attack and then gives multiple options for how to achieve this in its attack children. This relationship is often visualized through the lack of the vertical line between children that is present in the AND refinement. An example structure can be found in Figure 2.

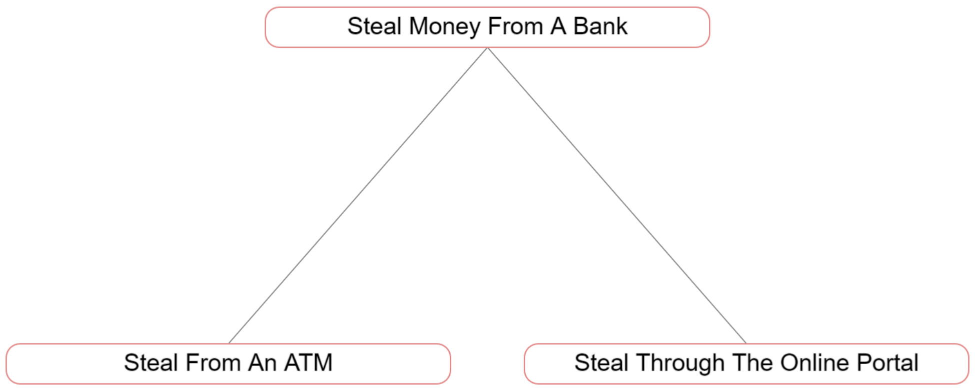


Figure 2: An example ADTree showing an attack node with the OR-refinement, and its children. The children describe options: you can steal from someone's bank account through an ATM *or* through the online portal of a bank.

Counter-measures are nodes that have a parent of a different type than themselves. An attack node with a defense node child calls that defense node child its counter-measure. Every node can have at most 1 counter-measure. Counter-measures describe an action that can be taken to counter the parent action. Examples of this can be found in Figures 3 and 4.

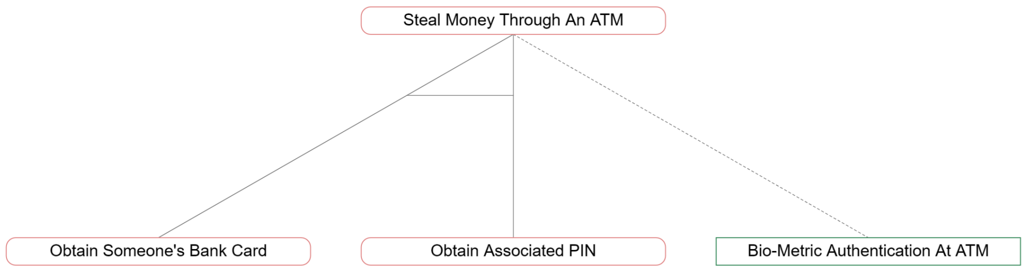

Steal Money Through An ATM
Obtain Someone's Bank Card
Obtain Associated PIN
Bio-Metric Authentication At ATM

Figure 3: An example ADTree showing an attack node with a counter-measure.

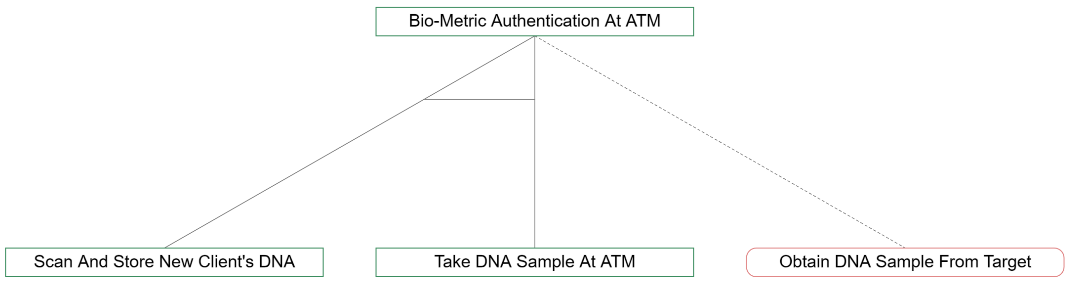

Bio-Metric Authentication At ATM
Scan And Store New Client's DNA
Take DNA Sample At ATM
Obtain DNA Sample From Target

Figure 4: An example ADTree showing a defense node with a counter-measure.

ADTrees can be expanded to contain cost values, probabilities and many other considerations, but those are outside of the scope of this thesis. More information about this and more can be found in various other papers [20, 21, 22], described in chapter 3.

## 2.2 Human Computer Interaction

The discipline of Human-Computer Interaction (*HCI*) seeks to study and understand the way that humans interact with computers. Due to the meteoric rise in computer use among people world-wide, this field of study has also increased in popularity and importance. In this paper we will make use of design principles and validation techniques outlined in the current body of literature regarding HCI. These principles were applied for eliciting requirements for the interface, within the theoretical framework of the project.

HCI seems at first glance like a very technical field, but Edmonds [2] states: ”As is well known to HCI practitioners, there is no simple cookbook of recipes for interaction and experience design. Rather, there are research and evaluation methods that involve users as part of the design process.”

Fallman [3] argues that the discipline of Human-Computer Interaction should be considered a design-oriented field, directed towards innovation, design and construction of new kinds of interaction technologies. He further argues that sketching in particular is a highly important and often underappreciated part of this design process. Within the process of creating the ADTWebApp, this was a very important inclusion: using (hand and digital) sketching as a way to iterate upon the conceptual design of the ADTWebApp, before programming work ever began. Allowing us to iterate not upon data structures and algorithms, but the very way that the user will interact with the application - designing the experience of the application, not the back-end. The use of sketching and this iterative process is further discussed in Chapter 4.

Some particularly important considerations come from Lewis [11] in which he discusses controversies from the very long history of HCI research. He compiles a list of advice that - as this project unfolded - turned out to be very apt and useful. For example, he states: ”Because “magic number” rules of thumb for sample size requirements for usability tests are optimal only under very specific conditions, practitioners should use the tools that are available to guide sample size estimation rather than relying on “magic numbers””. These finding were applied when trying to decide how many people to have participate in the experiment used to compare the 3 different versions of ADT-tools. The choice to include 9 participants was made primarily due to it being divisible by 3, while still leaving enough participants per tool to give enough data to compare between respondents of a single tool, aswell as being a reasonable scale for this paper. There is no commonly agreed upon 'minimum' for the amount of respondents, but we avoided any 'magic numbers' for this purpose anyway [11]. For the most part, this was a choice of convenience.

Lewis also introduces some important definitions which guided a lot of the research done: he mentions the two major conceptions of usability, ”summative” and “formative” usability. Usability's relation to HCI, as well as this distinction were important to our research after finding this source. We focused on formative usability: ”the detection of usability problems and the design of interventions to reduce or eliminate their impact (i.e., diagnostic usability)” [11]. As opposed to summative usability: ”the focus of which is on metrics associated with meeting global task and product goals (i.e., measurement-based usability)” [11].

Shneiderman [17] in the book *Designing The User Interface* presents 8 golden rules of interface design:

1. Strive for consistency
2. Cater to needs of diverse users
3. Offer informative feedback
4. Design dialogs to yield closure
5. Prevent errors
6. Permit easy reversal of actions
7. Support internal locus of control
8. Reduce short-term memory load

It is these principles that guide the design of the new interface. With the primary goal of the new interface being intuitiveness and accessibility regardless of user skill, rules 1, 2, 3, 5 and 8 are particularly important to consider in every design decision.

## 2.3 Requirement elicitation methods

### 2.3.1 Sketches

Sketches are simple drawings of designs that allow designers to ground ideas into reality, without committing to large scale designs. Their use was inspired by their similarity to Minimal Viable Products (MVP)s, which are common in software development, especially for smaller organisations.

Minimum Viable Products (MVPs) are a version of a new product with minimal included features, which allows a team to collect the maximum amount of validated learning about customers with the least effort. This definition is derived from the findings in Lenarduzzi [10]. MVP inspired the inclusion of sketching into the design process, to help take an ephemeral idea like - ”Let's make an ADT design tool” - and start grounding it in reality, helping to create requirements that are more specific than only asking for some amorphous tool to be created.

### 2.3.2 Mock-ups

Mock-ups are digital design illustrations, that are either non-functional or very minimally functionally. Tools such as Figma allow for basic transitions and interaction, but no actual functionality.

Mock-ups serve a similar purpose as sketches do: grounding ideas in reality. Mock-ups do this to a greater degree still, pushing the design more and more from ideas to real requirements that are actionable and definable. These mock-ups enable early assessment of both the cost and the value of the interface, this reduces uncertainties in the development process and allows for choices to be argued with 'user-experienced' information [12]. Mock-ups also align with the fail-fast concept of MVPs, giving early results and understanding, without large investments of time or money. These results will be more precise than in sketches, allowing even early hands-on testing to be done with any stakeholders.

### 2.3.3 Personas

A Persona is "a hypothetical archetype of a real user, which describes a user's goals, skills, and interests" [4]. The use of personas is valuable due to the unavailability of large scale and repeated validation techniques, like interview or questionnaires, due to time constraints. Without access to those techniques, the use of personas was a valid alternative. Using the PATHY technique proposed in Ferreira [4], we created personas that could be used to obtain further requirements for the final interface

## 2.4 Previous interfaces

### 2.4.1 ADTWebApp - previous interface

The old interface of the ATDWebApp is divided into a sidebar on the left of the screen, and a quick actions bar at the top of the canvas. The sidebar contains three tools (and one non-functional tool), as well as three miscellaneous tools under the "Other Tools" header.

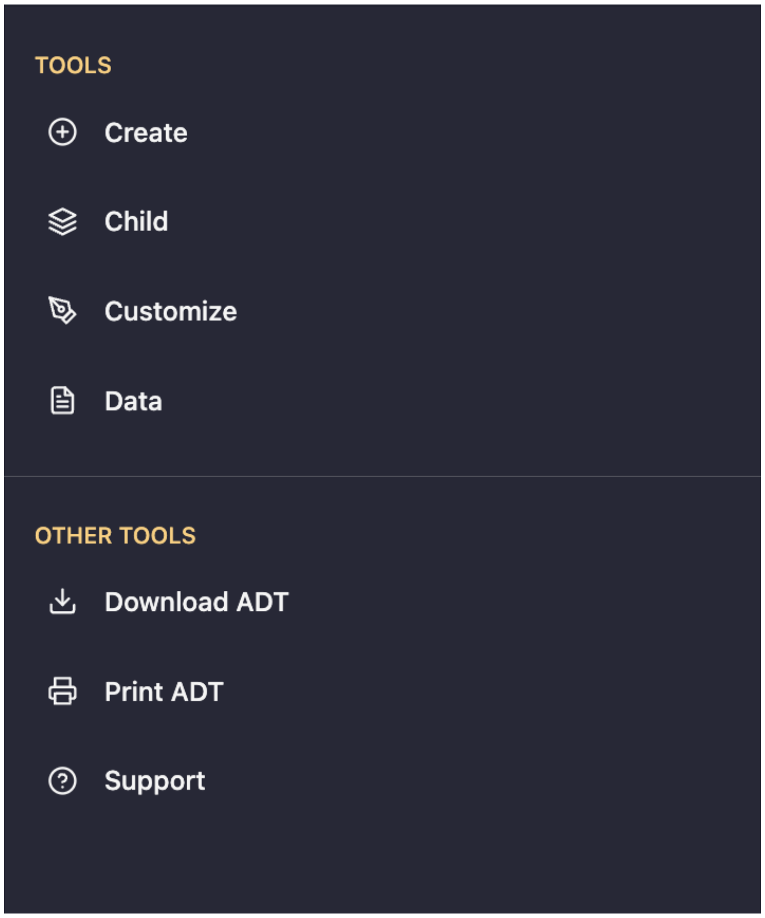
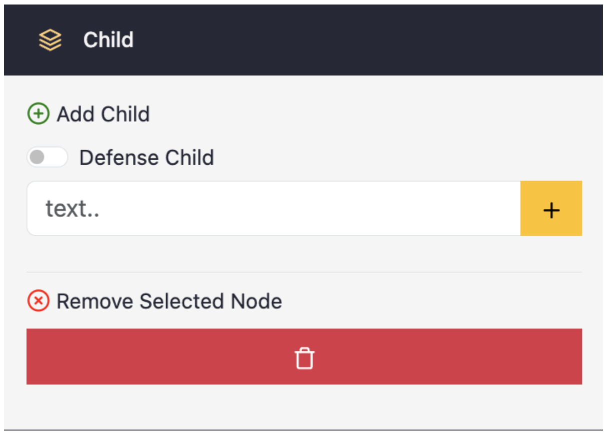
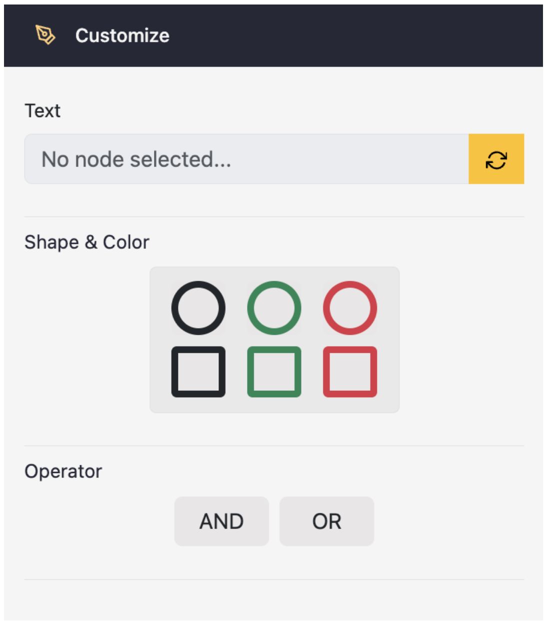


Figure 5: The entire sidebar of the previous interface.

Two of the main tools of the sidebar, Child and Customize, contain all node-specific functionalities. This interface achieves requirement 3 about as well as the new interface, however it lags behind in terms of efficiency of use, complexity and intuitiveness. By splitting the functions into two tabs, additional complexity is added, and efficiency of use is reduced. Having this be in a sidebar that is further away from the selected node also reduces efficiency, requiring further movement of the cursor to use the functionalities of the interface. All in all, the presence of a side-bar increases the short-term memory load. This goes against rule 8 as presented by Shneiderman [17].



(a) The Child tab of the sidebar, containing features used to create and remove nodes.



(b) The Customize tab of the sidebar, containing features used to modify selected nodes.

Figure 6: The two tabs that contain all node-specific functionalities.

Having functionalities that are all applicable only to a selected node be split into two tabs is also somewhat unintuitive and increase the short-term memory load, going against the 8th golden rule. The use of clear labels and selections does fulfill requirement 3, arguably better than the new interface.

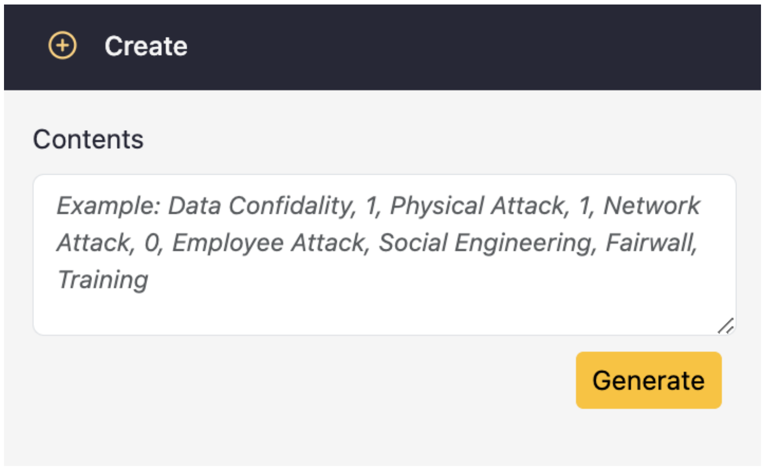

Create
Contents
Example: Data Confidality, 1, Physical Attack, 1, Network Attack, 0, Employee Attack, Social Engineering, Fairwall, Training
Generate

Figure 7: The Create tab of the sidebar, allowing users to generate trees using ADTLang.

The sidebar also contains the Create tab, which is not used as often as the graphical user interface for creating and modifying trees. Generally, the feature is only used to import a tree without the use of XML or JSON. As such, its inclusion in the sidebar takes up a lot of space, increasing complexity and intimidating users that are unaware of ADTLang, making it more difficult to learn for beginners. As such, this inclusion hampers fulfillment of requirements 2 and 3.

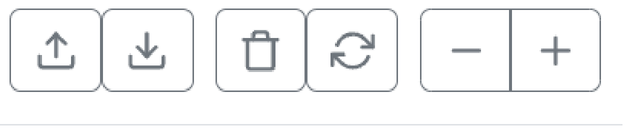


Figure 8: The quick actions bar of the previous interface.

The quick actions bar is largely the same across the new and old versions, only lacking the server interactions - since those has not been implemented yet, image download - located in the sidebar, ADTLang generation - located in the sidebar, and auto-formatting.

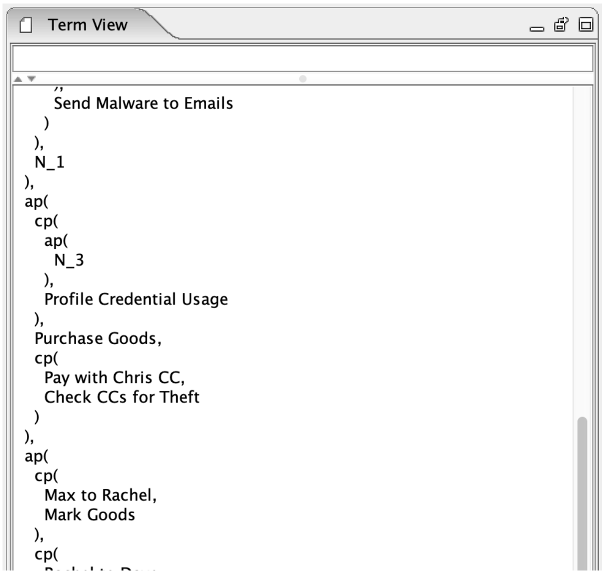
### 2.4.2 ADTool2

The interface of ADTool2 is vastly different from both ADTWebApp interfaces. It consists of 6 windows. (1) The Term View contains a code description of the tree. (2) The Tree View contains a visualisation of the tree, similar to the ADTWebApp's. (3) The Message Log contains a history of the manipulations made to the file, such as the creation and loading of trees and domains. (4) The Valuations View contains evaluations of created Domains. (5) The Ranking View ranks the top attacks of the selected Domain. (6) The Details View displays the details of the selected Domain.

ADTool2's interface does not achieve all 5 requirements to the same degree as both the new and previous interface of ADTWebApp. The code view is very unintuitive, has a large learning curve, adds a lot of complexity for simple modifications, and is very inefficient when creating trees. The tool generally also requires more setup to use, not just setting up windows, but also the application in general. Requiring the user to download the application, and also requiring the installation of JDK 7 (or later), adds more required setup to begin using the tool than the ADTWebApp does.

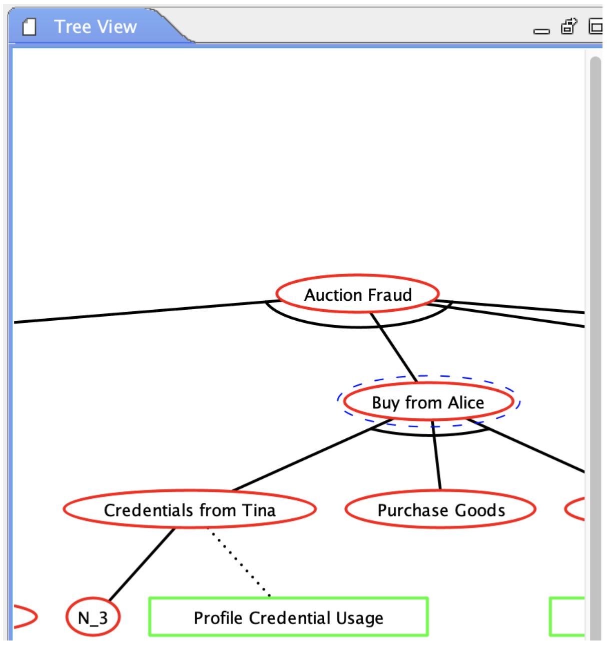
ADTool2 also enables the user to modify the tree by right clicking nodes, as opposed to using the Term View. This does far better when it comes to requirements 1, 2, 3 and 5. However, it is still less efficient than the new interface, requiring extra clicks to performs the functions of the interface.

ADTool2 features a lot of features that ADTWebApp does not contain, however even these features are not easily accessible. Bringing up the log, rankings, details and valuations views requires the users to activate them within the view menu. This adds extra complexity and is rather inefficient. Besides that, each view takes up extra screen space. This is somewhat compensated for by allowing the user to organise the views on their screen as desired, but any orientation of 6 views will become quite crowded.

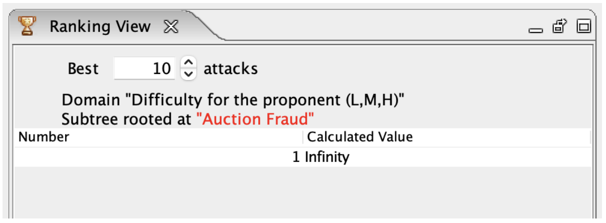


```
    Send Malware to Emails
   )
  ),
  N_1
 ),
 ap(
  cp(
   ap(
    N_3
   ),
   Profile Credential Usage
  ),
  Purchase Goods,
  cp(
   Pay with Chris CC,
   Check CCs for Theft
  )
 ),
 ap(
  cp(
   Max to Rachel,
   Mark Goods
  ),
  cp(
```

(a) Term view



(b) Tree view

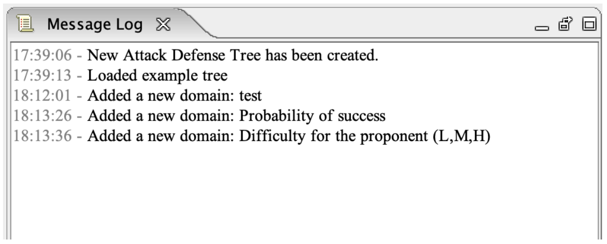


Best 10 attacks

Domain "Difficulty for the proponent (L,M,H)"
Subtree rooted at "Auction Fraud"

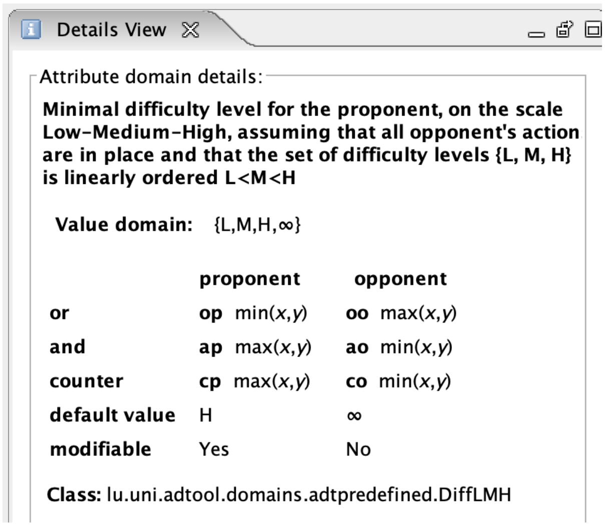
| Number | Calculated Value |
|---|---|
| 1 | Infinity |

(c) Rankings view



```
17:39:06 - New Attack Defense Tree has been created.
17:39:13 - Loaded example tree
18:12:01 - Added a new domain: test
18:13:26 - Added a new domain: Probability of success
18:13:36 - Added a new domain: Difficulty for the proponent (L,M,H)
```

(d) Log view



Attribute domain details:

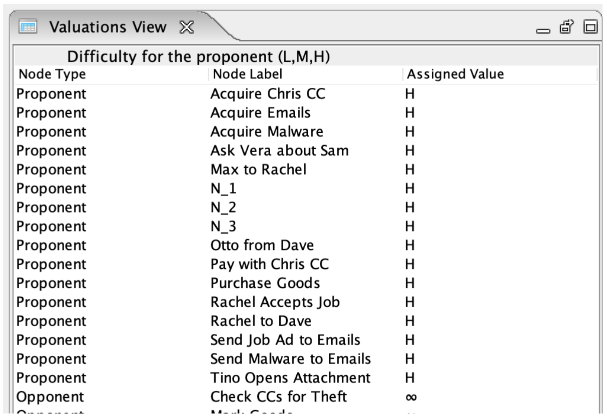
**Minimal difficulty level for the proponent, on the scale Low–Medium–High, assuming that all opponent's action are in place and that the set of difficulty levels {L, M, H} is linearly ordered L<M<H**

**Value domain:** {L,M,H,∞}

| | **proponent** | **opponent** |
|---|---|---|
| **or** | **op** min($x$,$y$) | **oo** max($x$,$y$) |
| **and** | **ap** max($x$,$y$) | **ao** min($x$,$y$) |
| **counter** | **cp** max($x$,$y$) | **co** min($x$,$y$) |
| **default value** | H | ∞ |
| **modifiable** | Yes | No |

**Class:** lu.uni.adtool.domains.adtpredefined.DiffLMH

(e) Details view



Difficulty for the proponent (L,M,H)

| Node Type | Node Label | Assigned Value |
|---|---|---|
| Proponent | Acquire Chris CC | H |
| Proponent | Acquire Emails | H |
| Proponent | Acquire Malware | H |
| Proponent | Ask Vera about Sam | H |
| Proponent | Max to Rachel | H |
| Proponent | N_1 | H |
| Proponent | N_2 | H |
| Proponent | N_3 | H |
| Proponent | Otto from Dave | H |
| Proponent | Pay with Chris CC | H |
| Proponent | Purchase Goods | H |
| Proponent | Rachel Accepts Job | H |
| Proponent | Rachel to Dave | H |
| Proponent | Send Job Ad to Emails | H |
| Proponent | Send Malware to Emails | H |
| Proponent | Tino Opens Attachment | H |
| Opponent | Check CCs for Theft | ∞ |

(f) Valuations view

Figure 9: The 6 available views within ADTool2.

## 3 Related Work

Two notable tools exist that facilitate user interaction with ADTrees: ADTool and its successor, ADTool2. ADTool features a straightforward user interface and boasts key features that include "...easy creation, efficient editing, and automated bottom-up evaluation of security-relevant measures." [8].

ADTool was later succeeded by a more advanced version known as ADTool2 [6]. ADTool2 enhances the original tool with several additional features, such as Sequential conjunct refinements in attack trees, Ranking attack trees, and Scripting capabilities. Although the interface of ADTool2 remains largely consistent with that of ADTool, due its increased functionality it was exclusively used in the comparison and evaluation of the final interface due to its advanced capabilities and improved usability.

Despite the similarities in interface design, ADTool2 provides notable usability improvements over its predecessor. It introduces several enhancements including the ability to copy and paste (sub)trees, manage multiple trees simultaneously, reorder child nodes, and efficiently handle particularly large trees.

Furthermore, ADTool2 has been extensively researched, which includes a case study conducted by the creators of the tool themselves [5].

The subsequent paragraphs explore related works concerning ADTrees. During the development of the interface, particular attention was given to incorporating features that would enable efficient integration of these advancements.

Tamjidi [20] explores the enhancement of the existing attack-defense tree framework by incorporating a dataset that includes organizational business processes, security data, business assets, vulnerabilities, and associated security countermeasures. This inclusion aims to facilitate more informed and effective defensive strategies. When developing the interface, the potential integration of these features was a priority, ensuring that their addition would not necessitate substantial modifications to the existing code.

Wang [21] introduces an improved version of the ADTree, referred to as iADTree, which focuses on determining the most effective defense policy for countermeasure selection along each attack path. This version employs probabilistic analysis for each node, which aids defenders in simulating attack outcomes. By prioritizing more critical attack paths over less significant ones, this approach supports better defensive planning. The interface development took this into account, allowing future developers to incorporate such simulations with minimal changes to the existing codebase.

Xu [22] presents two methods for selecting optimal defense nodes to counteract specific attacks while minimizing costs. The first method introduces the atom attack defense tree (A2DTree), which imposes various constraints on a conventional ADTree. The second method employs an algebraic approach to efficiently calculate the minimum cost of defense, leading to the creation of a minimum defense cost calculation tool. During the interface development, the feasibility of incorporating such a calculation tool was considered to ensure that its integration would not require significant alterations to the backend of the interface. However, enabling support for A2DTrees was not given the same level of consideration and would require a more extended implementation effort within the interface.

The System Usability Scale (SUS) [1] is a widely recognized and straightforward questionnaire designed to assess the usability of a system. Despite the fact that this research was first published back in 1996, the insights and findings from SUS continue to be highly relevant and frequently referenced in contemporary research. Its modern significance is made apparent by the many papers that still discuss the essential attributes and advantages of using SUS to evaluate usability. In our study, the SUS was employed to analyze and assess both the strengths and weaknesses of the various interfaces, providing valuable insights into their relative effectiveness and user satisfaction.

PATHY2.0 is a method that supports the development of persona descriptions that are targeted and informative [4]. Ideally when designing interfaces, user feedback would be incorporated into the process as early and frequently as possible. This is quite difficult however, since creating feature-rich enough prototypes to warrant usable feedback is expensive. The time spent organising moments for feedback and the money spent developing prototypes, both of which would be incurred multiple times over the course of development, make this approach prohibitively expensive for most developments. Personas are a way to understand the users' needs while not incurring the time and monetary costs associated with end-user feedback. Personas on their own may have lots of irrelevant information, which makes it more laborious to properly understanding the requirements from users. This was alleviated by using the PATHY2.0 framework when creating personas. Those personas were used to great effect within this thesis to elicit requirements.

# 4 Methodology

The previous user-interface of the ADTWebApp was created by Marwa [15]. A theoretical framework was already proposed by Marwa, which was then inherited when creating this project. The theoretical framework proposed herein takes the inherited work and modifies it to fit a new understanding of the user-interface, while also limiting its scope to what was created within this project. For example, considerations about the web-app's navigational bar are omitted, since those were not considered nor modified within this work.

## 4.1 Applied process

We applied the following process as pictured in Figure 10 when designing the ADTWebApp's new user-interface.

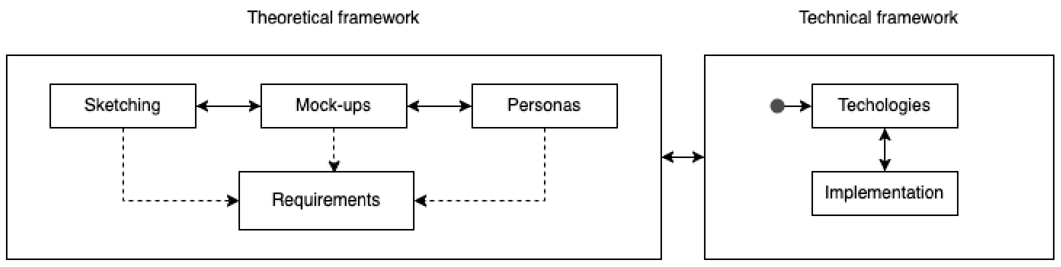


Figure 10: A visual representation of the used method.

This process consisted of two phases, the first of which is the theoretical framework, in which the primary objective was to construct a list of requirements for the user-interface. This was done by creating sketches, mock-ups and personas to better understand the interface and what it should provides users, before having to create it.

The second phase is the technical framework, where the technical details of the implementation of the user interface are decided (such as frameworks) and executed upon by implementing the user interface.

Throughout this process, we moved between phases and between parts of phases. Creating personas, only to realise new mock-ups are needed, for example.

## 4.2 Theoretical framework

The theoretical phase contains any requirement elicitation steps taken that help designers compile a list of requirements. Each project should have its own chosen steps to help aggregate these requirements, but the proposed model contains some steps that are inspired by general requirement elicitation, HCI principles and software development concepts.

The first of these requirement elicitation steps is sketching the interface. Sketching is an important part of the HCI process [3], whose inclusion is also inspired by the concept of Minimum Viable Products (MVP), often found in software development - particularly in small startups. These sketches were used to create a more grounded design before any development has occurred, from which early requirements can be derived.

The second elicitation step taken is the creation of mock-ups. Mock-ups serve a similar purpose as sketches do: grounding ideas in reality. These allowed us to elicit more defined and actionable requirements for the interface.

The third elicitation step used is the use of Personas. Using the PATHY technique proposed in Ferreira [4], we created personas that could be used to obtain further requirements for the final interface.

The interface has multiple functionalities, all of which must work with the definition and rules of an ADT. These functionalities must also be user-friendly and intuitive. Marwa [15] defined a set of use-cases which were designed and implemented for the previous interface. A new set of use-cases was defined, showing the way in which the user will be able to interact with the new interface.

## 4.3 Evaluation

Evaluating the quality of the interface, and comparing its quality to those of the other ADT-tools, was a 3 step process. This process was used to understand where improvements had been made, where the interface had gotten worse, and where improvements were still required.

Firstly, we gave the participants some time to simply use the tool. Before beginning, they were given an overview of the purpose of the tool, and its target audience. They were told the tool is meant to model (cyber-)security scenarios, meant to be used by not just cybersecurity specialists, but also by students, managers and anyone else that might wish to analyse or model security scenarios. They were informed that the tool starts up with an example scenario and were then asked to simply create a scenario from scratch, until they felt they had explored the entire tool and made something they were satisfied with. Upon deciding they were done, a picture was taken of their scenario and saved.

Secondly, the participants were given a questionnaire. For this questionnaire, the System Usability Scale (SUS) [1] was used. A totally custom questionnaire was considered, however using standardized questionnaires is generally preferred [11]. The participants were clearly informed that their ratings on the questionnaire would not cause the surveyors any academic harm or otherwise. So giving 'negative' answers was not discouraged or harmful, and 'positive' answers were not encouraged or beneficial. The only data collected on the participants was whether they were in computer science related fields or not, without any identifiable data to go along with it. All of them agreed to their responses and this limited data about them being collected for the purposes of this research. None of the participants were compensated for their participation.

Lastly, participants were given the opportunity to explain their answers on the questionnaire, as well as discuss their experience with the program. During this interview, we simply recapped the parts of the evaluation, and went over their answers, asking questions to better understand the feelings of the respondents regarding the tool's interface.

The System Usability Scale consists of 10 total question, each of which can be answered by choosing on a scale from 1-5, with 1 being "Strongly disagree" and 5 being "Strongly agree":

1. I think that I would like to use this system frequently
2. I found the system unnecessarily complex
3. I thought the system was easy to use
4. I think that I would need the support of a technical person to be able to use this system

5. I found the various functions in this system were well integrated
6. I thought there was too much inconsistency in this system
7. I would imagine that most people would learn to use this system very quickly
8. I found the system very cumbersome to use
9. I felt very confident using the system
10. I needed to learn a lot of things before I could get going with this system

A total of 9 participants took part in the experiment. They were divided over the 3 following ADT-Tools: ADTool2 [8], the previous interface of the ADTWebApp, and the current ADTWebApp interface. ADTool2 was chosen for comparison since it is a particularly modern tool for ADT creation that has had multiple papers written on it, including two demonstrations [8, 6] and a case study [5].

Of the 9 participants, 6 were either Computer Science students, or from Computer Science adjacent courses such as (Business-)IT. The remaining 3 did not have a computer scientific background.

# 5 Implementation

Initially, the only added use-case required was the ability to move nodes around the screen. We initially preferred that the original sidebar remain in place and functional, following discussion with key stakeholders. Our interpretation of the to-add functionality, the given preferences, and what was to remain of the original interface, drastically changed over the course of development. By applying multiple HCI design techniques and requirement elicitation techniques, our understanding of the interface became clearer, until eventually development could begin. These findings will be discussed chronologically in the following subsections.

The existence of the previous interface was quite a boon to the development process, allowing requirements for the new interface to be garnered by analysing the previous interface. It was quickly apparent that the side-bar took up a lot of space, making the canvas size quite limited.

As such, one of the first expansions we made upon the interface was that simply being able to move nodes was not enough of an improvement to the interface, although that realisation mostly came from intuition. It was only once mock-ups and personas were applied that the underlying reasoning became clearer.

## 5.1 Sketches

All sketches made have an implied ability to move the nodes, most of the changes were made to the surrounding interface instead. The first two sketches keep the original design of the interface somewhat intact: a menu and a canvas. The canvas allows you to select nodes, and the menu allows you to make modifications to the selected node, just as in the previous interface.

In the first iteration, it was attempted to increase the size of the canvas by simply hiding the sidebar whenever it was not in use. The sidebar itself would contain all of the modifications that can be made to a node, such as changing its content, attack/defense type, refinement or styling. It would allow the user to add a child node to the currently selected node or remove the selected node. Presumably in some future expansion of the project, probabilities, costs and other factors could be implemented in this sidebar too.

All of the non-node interactions - such as generating a tree using ADTLang [7], downloading the ADT as an XML file or downloading the ADT as a JPG image - would be available in the small quick actions bar at the top of the canvas.

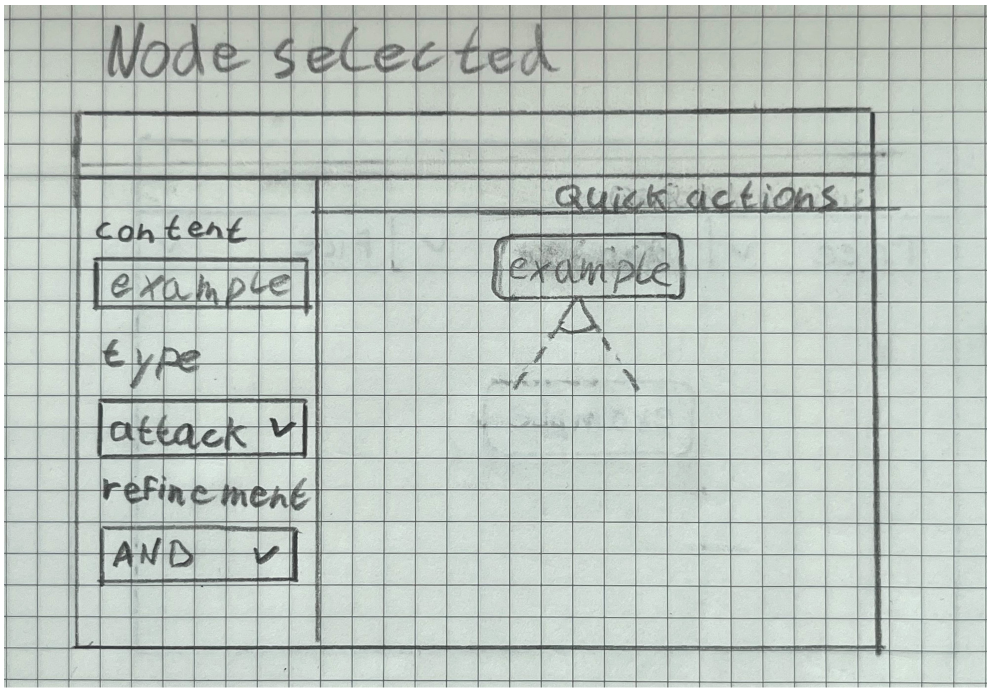


Figure 11: The initial sketch of the interface with a dynamic sidebar, which is revealed when a node is selected to be manipulated.

From this sketch it became apparent that both the sidebar and the quick actions bar would become quite crowded. As such, the next iteration of the sketch contains two sidebars. The left sidebar contains the node-specific functionalities: editing content, changing type, changing refinements, adding children, removing the node or applying styling. The quick actions bar would be removed and its functionalities, including the functions that were in the left sidebar in the original interface, would be implemented in the right sidebar. This sidebar would be able to give each function more space, while also allowing the entire sidebar to be hidden away, freeing up more space on the canvas.

This split interface introduced a new learning obstacle: new users would have to learn where the functionalities reside, and forgetting where a specific functionality is might become frustrating over time. Besides that, introducing another click to reveal them adds friction to the user's experience, especially since the two sidebars are on entirely opposite ends of the screen.

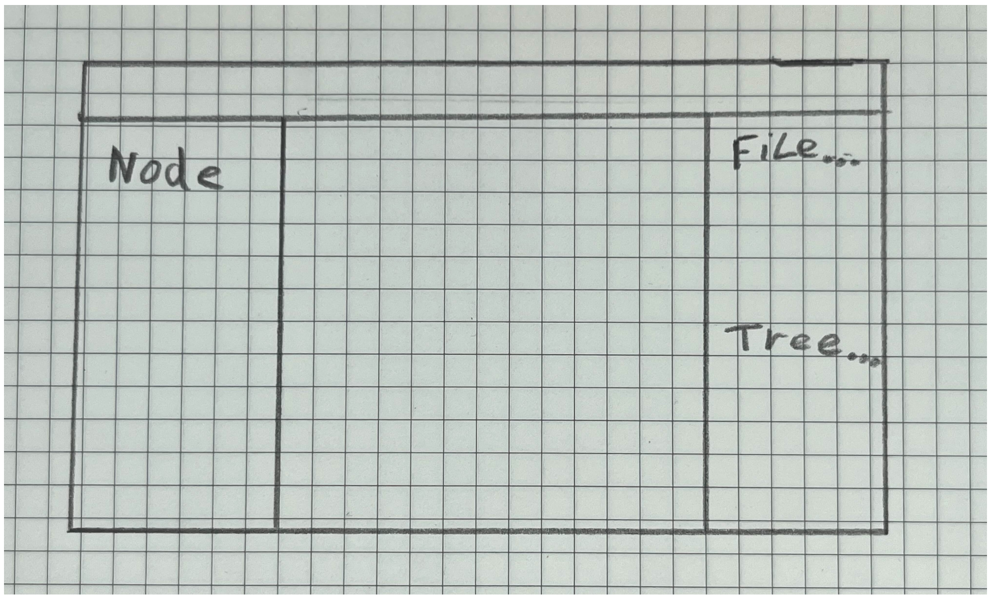


Figure 12: The second sketch of the interface, with 2 sidebars. Each of which can be manually collapsed.

To avoid having things scattered through-out the interface, a sketch was created in which all of the menu items were sorted into one of three categories on a single bar. The Tree menu contains all items that modify the tree, such as generation using ADTLang, subtree deletion, auto-formatting and downloading the tree as an XML or JPG. The Node menu contains the content editing, type selection, attack-defense selection, refinement and styling. The File menu contains all options pertaining to the back-end of the web-application, such as uploading the ADT to the server, and also retrieving from the server at a later date.

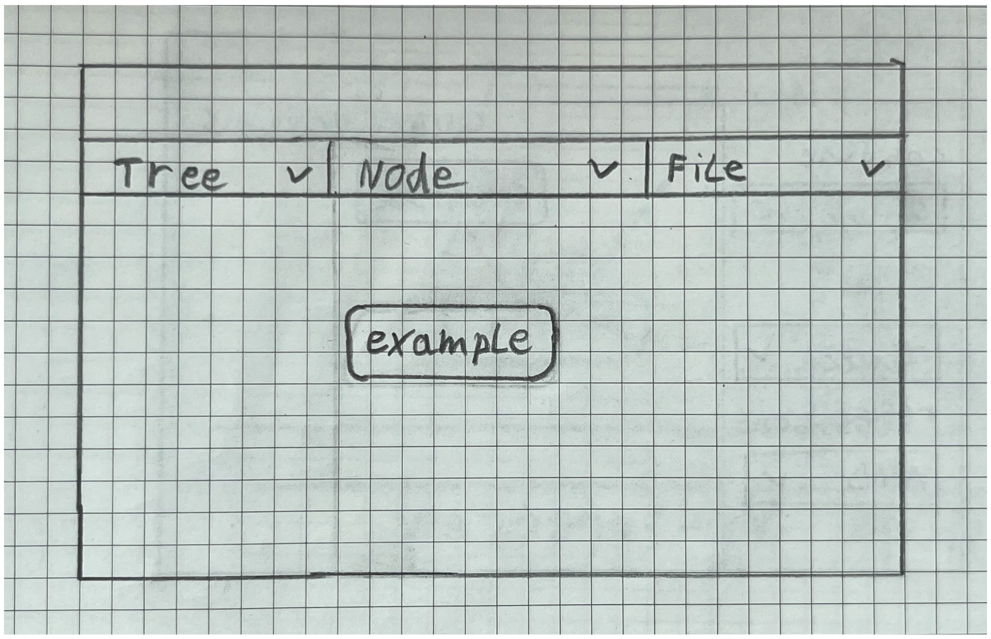


Figure 13: The third completed sketch of the interface, which forgoes the sidebar entirely in favor of a single dynamic toolbar.

The exact menu item lay-out could change, but this design achieves two goals: it expanded the canvas space a lot, while keeping everything quite accessible. However, achieving this vision would require a large overhaul of the existing interface. As such, it was decided that the interface as pictured in the first sketch (Figure 11) should be used going forward, to limit the technical requirements of the project.

## 5.2 Mock-ups

The first mock-up was based on the first sketch pictures in Figure 11. This mock-up features a single, dynamic side-bar, which is expanded when a node is selected, and collapsed when no nodes are selected.

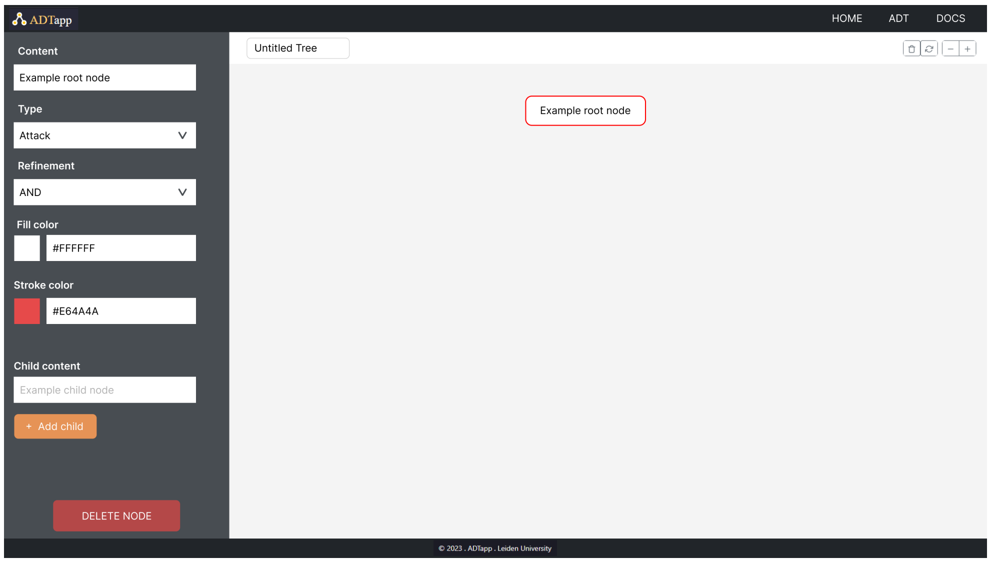


Figure 14: The mock-up based on the first sketch made, with the sidebar extended. The selected node is highlighted and its settings are visible in the sidebar.

When expanded, the side-bar features all of the important functions for a node: editing its contents, type and refinement. It also features the ability to edit its styling by changing the fill and stroke colors. A child can be added, with some set content, and the selected node can be entirely deleted. The features in the mock-up are implied, and supposed to show the way that the interface could look after implementation, without implementing any of the features.

Upon deselecting a node, by clicking anywhere that is not a node, the side-bar collapses. The side-bar takes up 25% of the screen in this mock-up, so this expands the available canvas size by 33%.

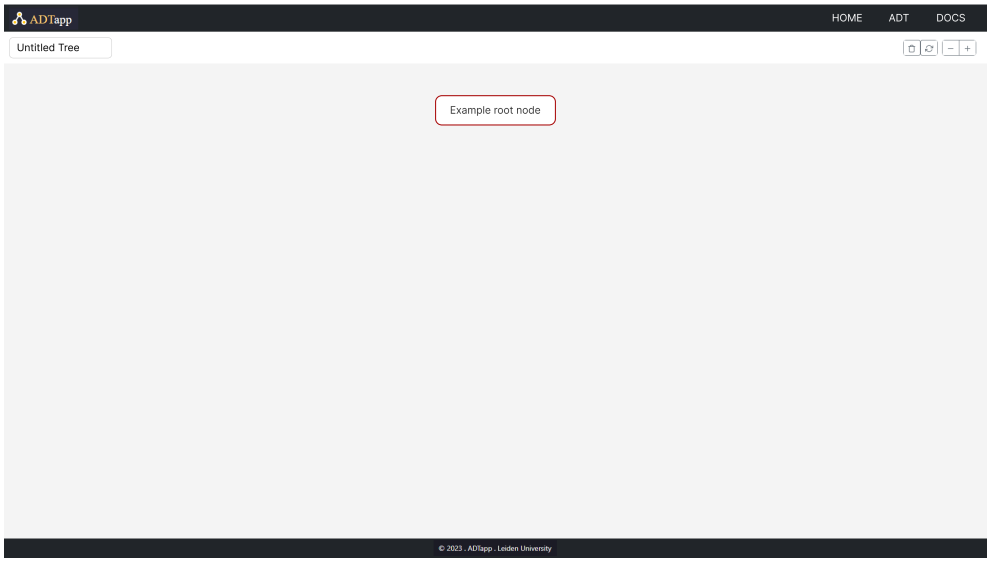


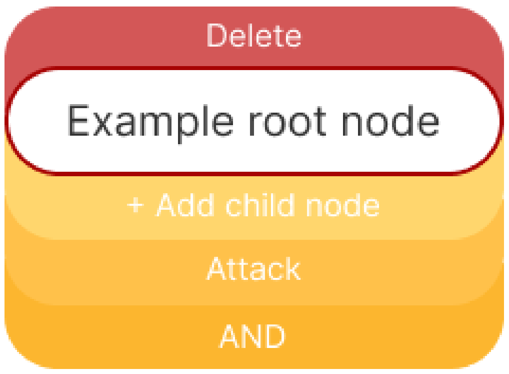
Figure 15: The mock-up based on the first sketch made, with a collapsed sidebar. No nodes are highlighted.

This mock-up was sufficiently able to achieve all of the basic requirements that were recognised at this point chronologically: maintain the original sidebar (as requested by key stakeholders), expand the available canvas space - as intuited based on the previous interface, and adding the ability to move nodes.

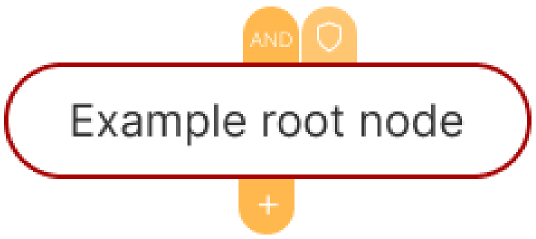
After using personas to elicit additional requirements, however, it became apparent that this interface was good, but not good enough to meet the additional requirements. These additional requirements are discussed and explained in Chapter 5.3. The garnered requirements are the following:

1. The interface should facilitate a quick and efficient ADT creation process.
2. The interface must not be complex.
3. The interface must be able to be quickly learned by non-technical users.
4. The interface must not require any complex setup to use.
5. The interface must be intuitive.

Based off of these requirements, the mock-ups in Figure 16 were created. These mock-ups do away with the sidebar(s), and instead move all of the node-specific functionalities to the select node. This serves to achieve requirements 1, 2 and 5.



(a) The first mock-up, in which the select node's options are given using text.



(b) The second mock-up, in which the selected node's options are given using iconography.

Figure 16: Two trialed versions for the node interface, based on newly acquired requirements.

The first mock-up aims to retain the spirit of the original side-bars, being very clear in what each function does by using text to describe the options. Although this is a simplification from having to type content and select options in the sidebar, this interface is still somewhat cluttered. The second mock-up simplifies this concept even further, using iconography to represent the ability to add a child node, and toggle attack-defense types. In Figure 17 this node design is visualized within the larger interface, where the entire sidebar is omitted.

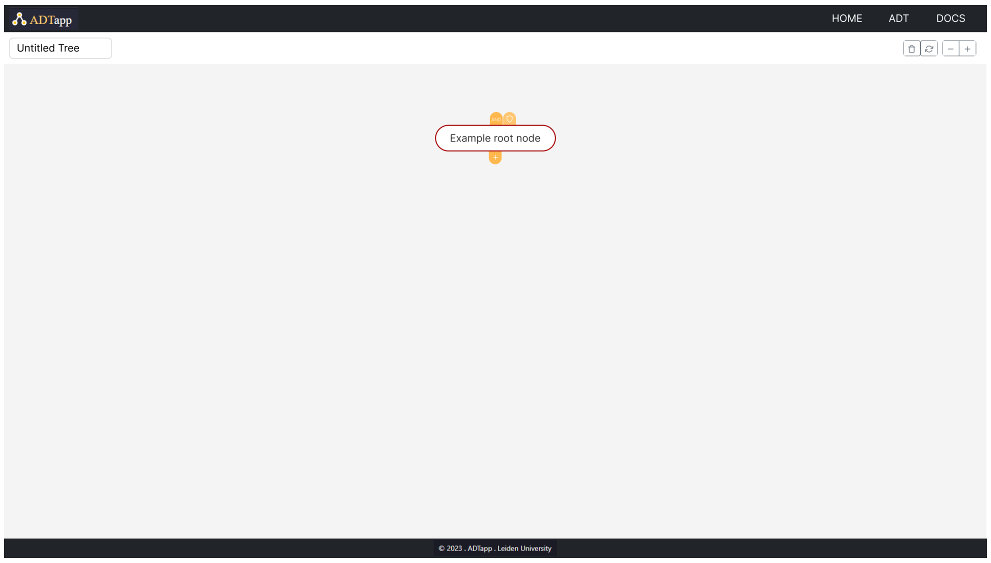


Figure 17: The mock-up based on the new requirements, without a sidebar.

Using the tool becomes a lot more efficient, simply due to having to move the cursor less far to manipulate nodes, which is the most frequently used set of features when creating ADTs. The tool also becomes less complex, since manipulating the nodes has become more straight forward. Rather than select the attack-defense type from a drop-down, or the refinement from a drop-down, you simply toggle them with a single click. Editing the contents of the node would be possible by typing directly into the node, rather than updating its contents within the sidebar, making the users actions more immediately visually apparent. This mock-up does forgo the ability to change the styling, although that could be added as one of the quick actions. The quick actions bar now contains all of the non-node features, like deleting the tree, downloads/uploads, server interactions and use of ADTLang.

## 5.3 Personas

Personas were used in order to better understand and garner further requirements. Three different personas were designed for this purpose, using PATHY2.0 [4]. For better formatting, the terms "Technology experiences" and "Existing solutions" were shortened to Experience and Solutions.

**Persona 1: The Student**

- **Who:**
  A 21 year old undergraduate studying Cybersecurity. Interested in technology, but often feels overwhelmed by complex concepts. Concerned about understanding real-world applications of theoretical knowledge.
- **Context:**
  Lives in a dorm, engages in group projects, often collaborating with peers. Does not have much time, being a full-time student. Procrastinates at times, and might not fully understand subject matter. Spends a lot of time outside of their dorm working on projects.

- **Experience:**
  Proficient with security tools and platforms such as Kali Linux and Wireshark. Prefers intuitive interfaces and visual representations, dislikes cluttered or overly technical tools. Likes having tools always be available.

- **Problems:**
  Struggles to visualize and understand complex attack scenarios. Needs an effective way to represent threats and defenses for assignments and projects, sometimes last minute or while not at home.

- **Needs:**
  A user-friendly interface - available anywhere - for creating attack-defense trees that allows for easy and quick visualization and manipulation of security concepts.

- **Solutions:**
  Current tools like Microsoft Visio, Figma or draw.io are too general and not specialized for cybersecurity applications. A dedicated tool for attack-defense trees is needed.

The Student heavily prioritizes tools that are not too complex and allow them to create ADTs quickly. Being a full-time student means that they might wish to create ADTs promptly, especially when a deadline is quickly approaching. Quickly being able to iterate on trees in a team and communicating scenarios is also valuable. They also value simplicity, while still still being specialized for ADTs. Not requiring a large investment to learn or setup is required. Three requirements can be extracted from The Student persona:

- The interface should facilitate a quick and efficient ADT creation process.
- The interface must be usable by users without the need to learn or set up anything difficult.
- The interface must be usable and available from multiple locations and devices.

**Persona 2: The Security Analyst**

- **Who:**
  A 38 year old security analyst working at a mid-sized company. Experienced but often feels pressured to present findings clearly and effectively. Concerned about communicating complex security concepts to stakeholders who might not be as technical.

- **Context:**
  Works in a fast-paced office environment, interacting with IT teams and management. Regularly participates in security assessments and incident response activities.

- **Experience:**
  Uses security analysis tools such as Splunk and various threat modeling applications. Values clear visualizations, dislikes tools that require extensive training or lack integration into other tools.

- **Problems:**
  Struggles to illustrate security threats and defenses to non-technical stakeholders. Needs a way to clearly map out attack-defense strategies for reports and presentations.

- **Needs:**
  An intuitive interface that enables the creation of detailed attack-defense trees for effective communication with stakeholders and team members.
- **Solutions:**
  Existing modeling tools either do not cater specifically to cybersecurity needs, or are not intuitive enough. Many also lack any real integration. A specialized solution that is intuitive and focuses on attack-defense scenarios is required, which works with other common security tools.

The Security Analyst values a tool that can quickly be used to effectively communicate with team members and stakeholders. Working in a fast-paced environment also means that being able to integrate the tool into their workflow and having it work with the rest of their tools is a very valuable asset. We extracted the following requirements from this persona:

- The interface must allow easily readable ADTs to be constructed intuitively.
- The tool must allow users to integrate the tool into their larger workflow.
- The interface must feature applicable cybersecurity analysis tools.

**Persona 3: The Incident Response Manager**

- **Who:**
  A 35 year old incident response manager who requires better collaborative tools. Concerned about timely and effective responses to security incidents.
- **Context:**
  Manages a remote team, coordinating efforts during security incidents. Frequently engages with multiple teams and external partners during crisis situations.
- **Experience:**
  Utilizes incident management tools like JIRA, and communication tools like Slack. Prefers collaborative platforms with good integration.
- **Problems:**
  Wants to develop and communicate attack-defense strategies during incidents. Needs a collaborative tool that allows her team to visualize threats and defenses effectively.
- **Needs:**
  A collaborative interface for designing attack-defense trees that facilitates teamwork and ensures everyone is on the same page during security incidents.
- **Solutions:**
  Current tools are either too complex or do not support real-time collaboration on attack-defense trees. A solution that combines usability with collaboration features is essential.

The Incident Response Manager values being able to effectively communicate and collaborate. The tools they use should be able to be used during urgent incidents to communicate to a large amount of people. From this persona we can extract the following requirements:

- The interface must allow real-time collaboration between multiple users.
- The interface must not be complex, and able to be quickly learned by multiple large teams.
- The interface should facilitate a quick and efficient ADT creation process.

From the nine total requirements, two are duplicate. Since both the Student and the Incident Response Manager face scenarios in which they must be able to quickly communicate threats and defenses, they both feature the requirements that the interface should facilitate a quick and efficient ADT creation process.

The requirement to be able to access your work anytime, anywhere - as extracted from the Student persona - was also implemented into the tool at this point by another participant in the project, meaning that this requirement only requires the new interface to carry over the existing feature.

The Incident Response Manager wants the tool to be collaborative, this is something for further research, and not within the scope of this thesis.

The Security Analyst wants the interface to feature security analysis tools, as well as integrate into their workflow. The applicable analysis tools, as discussed in Chapter 3, are potential subjects for future research, and not within the scope of this thesis. Neither is the integration of the tool, as that would require significant research into other security tools, and veers away from being strictly about interface design.

This leaves the following adapted, simplified requirements for the interface:

- The interface should facilitate a quick and efficient ADT creation process.
- The interface must not be complex.
- The interface must be able to be quickly learned by non-technical users.
- The interface must not require any complex setup to use.
- The interface must be intuitive.

## 5.4 Technologies

The ADTWebApp is entirely made of HTML, CSS and JavaScript. JavaScript is the primary language (99.2% according to GitHub), using the P5.js and JQuery frameworks. JQuery was primarily used for some of its quality-of-life features, but P5.js was used very intensively.

P5.js changes the way in which HTML elements are controlled through JavaScript, by adding its own abstraction layer over them [13]. These offer much simpler and user-friendly control to the programmer using them. One of these controls is the `position()` command, which allowed us to take input elements and move them anywhere on the screen [14]. This change enabled the ability to simply type into the node, rather than changing its content in an input box on the side, since the node itself was now a regular HTML element. The previous interface instead used a lot of P5.js's canvas tools to essentially create its own input elements. This, however, created lots of tech debt due to the bug-prone nature of making everything custom.

This change, combined with the recognition that a large overhaul of the interface - as well as removing the sidebar - was necessary for a better user experience, is what inspired the decision to start mostly from scratch. We removed all of the existing JavaScript code, save for a few lines, and began from scratch. P5.js's improved event handling, as well as the ability to position HTML elements anywhere, made the implementation of the node manipulation a lot simpler to implement.

The P5.js canvas object was used to make non-node portion of the tree; the lines connecting the nodes and the refinements.

JQuery was used mostly to help simplify some of the tedium of JavaScript, rather than using any of its core features, although that could be the subject of future research.

# 6 Results

After development of the new interface was mostly completed, a study was conducted in which we aimed to evaluate this interface. In the following sections, the new interface is shown and explained, and the results of the conducted study are shown with context given by participants.

The new interface was designed according to the following list of requirements:

1. The interface should facilitate a quick and efficient ADT creation process.
2. The interface must not be complex.
3. The interface must be able to be quickly learned by non-technical users.
4. The interface must not require any complex setup to use.
5. The interface must be intuitive.

## 6.1 Use-cases

Marwa [15] defined a set of use-cases, which were adapted to the new interface. Although mostly the same, a large structural change was made to the way that the user interacts with the interface. It has become responsive, specifically to the user selecting a node. This allows the user to select the various functions available to them, to apply to the selected node. This condensing of functionalities has caused the use-case diagram to undergo large changes.

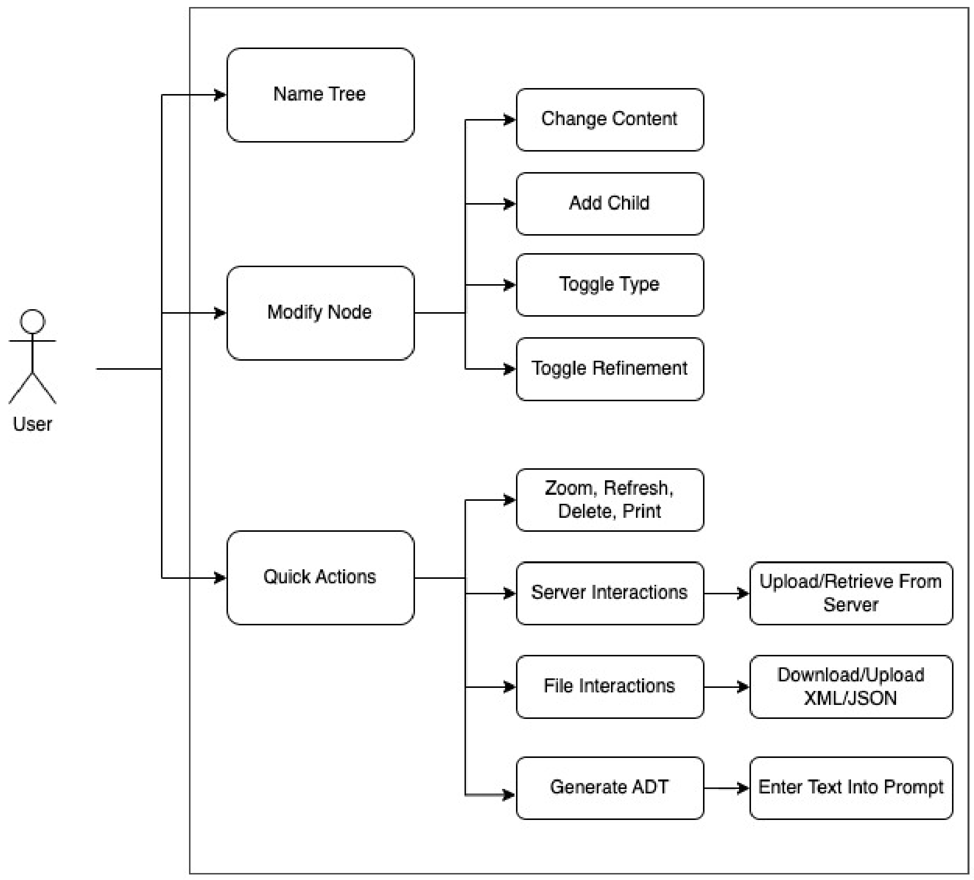


Figure 18: A visual representation of the use-cases.

The final interface's use-cases have been greatly compressed. Rather than having the node modification options be continually available, they only become available when a node is selected. As such the main use-case is the modification of a node, which then gives various options for its modification. Lots of options that were previously in the sidebar, have now also been compressed under the quick actions bar.

## 6.2 Interface

The current interface is divided into two sections: node controls and the quick actions bar. The node controls contain all of the functionalities of the interface that pertain to editing the ADTree itself. It consists of four buttons:

- Add child (+)
- Delete node (x)
- Toggle attack-defense (shield / crossed-out shield)
- Toggle refinement (AND / OR)

Besides the buttons, the contents of the node are edited simply by typing after selecting the node.

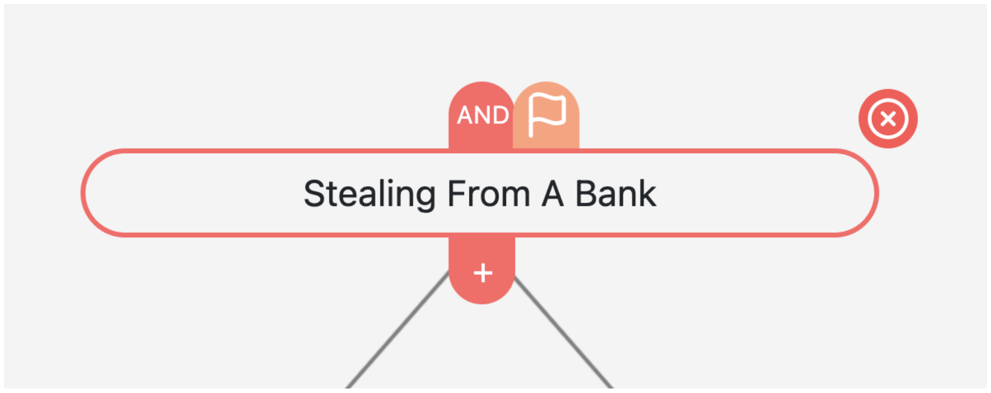
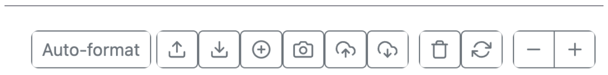


Figure 19: The node-specific controls of the new interface.

All of the other controls are combined into simple to access icons on the quick actions bar. On the left the ADTree can be named, and on the right the following quick actions are contained, listed from left to right:

- Auto-formatting
- Upload an XML/JSON file a tree
- Download an XML/JSON file of the tree
- Generate a tree using ADTLang
- Generate an image of the file tree
- Upload the current tree to the server
- Retrieve a tree from the server
- Delete the entire tree
- Refresh the website
- Zoom out
- Zoom in

(a) The left half of the quick actions bar, containing the tree's name.



(b) The right half of the quick actions bar, containing all non-node functionalities.

Figure 20: The two parts of the quick actions bar, shown separately due to the width of the bar.

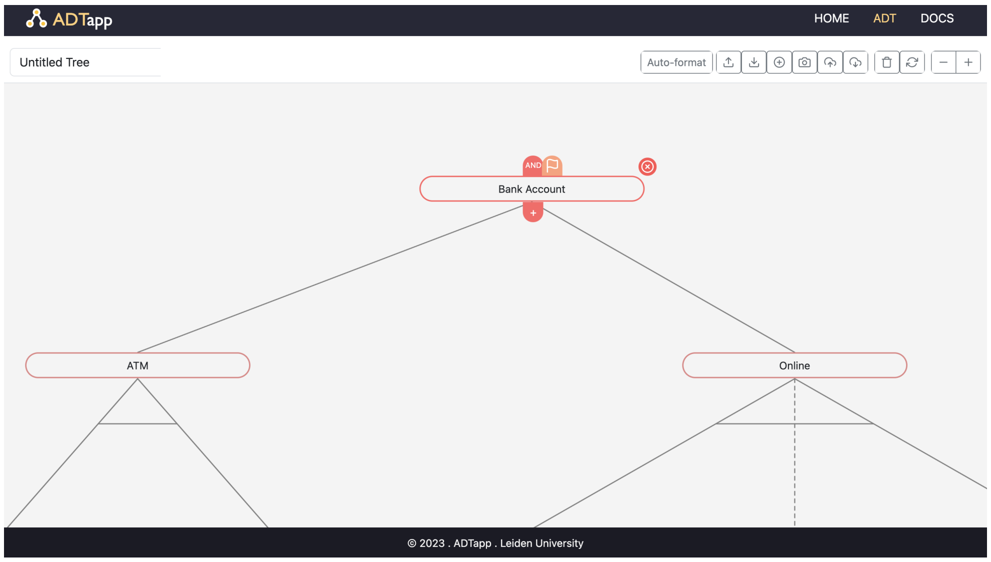
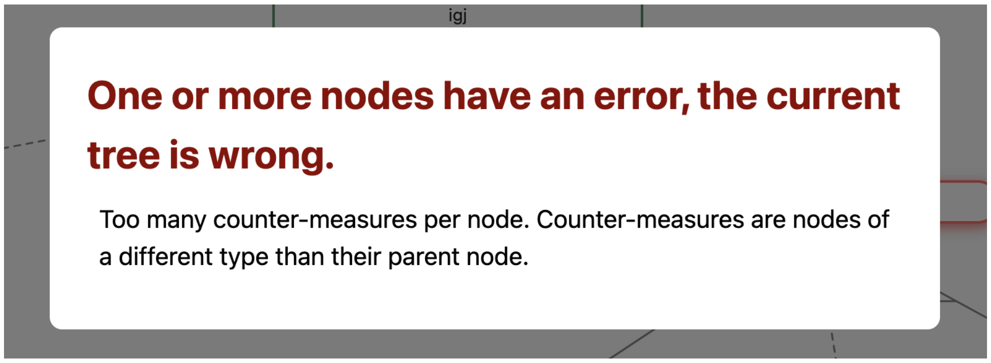


Figure 21: The entire new interface, displaying a simple example tree.

The interface also provides visual feedback to the user in the form of a warning at the top of the screen, when the current tree does not conform to all the rules of an ADTree.

Figure 22: The warning sign shown at the top of the screen when the current tree is not a correct ADTree.

Upon clicking on the warning, the user is informed of the error(s) that are present within their tree. Besides this, the nodes that are wrong are highlighted red, to show the user where mistakes have been made. These features are in accordance with golden rule 3, as presented by Shneierman [17].


One or more nodes have an error, the current tree is wrong.

Too many counter-measures per node. Counter-measures are nodes of a different type than their parent node.

Figure 23: The warning message shown to the user upon clicking the warning sign. Currently the only rule implemented is the counter-measure rule, as shown.

The node-specific controls were designed to achieve requirements 1, 2, 3 and 5. The available quick actions achieve requirement 4 by providing the user with options to generate a tree, import trees, store them remotely and even auto-format them. This ensures that the user does not need to do a lot of work to begin working with the tool, or continue working with the tool.

The inclusion of the warning does a lot to make the interface more intuitive, helping achieve requirement 5. It also helps achieve requirements 2 and 3, since simply using the tool will help users learn how to use it, rather than needing to learn everything upfront.

## 6.3 Evaluation

The evaluation process was divided over multiple days, with each interview being conducted individually. The interview consisted of a hands-on phase - which started with a simple introduction, followed by a questionnaire and then ended with some time to simply talk over their answers and experience. The following questions were asked during the questionnaire:

1. I think that I would like to use this system frequently
2. I found the system unnecessarily complex
3. I thought the system was easy to use
4. I think that I would need the support of a technical person to be able to use this system
5. I found the various functions in this system were well integrated
6. I thought there was too much inconsistency in this system
7. I would imagine that most people would learn to use this system very quickly
8. I found the system very cumbersome to use
9. I felt very confident using the system
10. I needed to learn a lot of things before I could get going with this system

The exact structure of the evaluation is explained in chapter 4.3.

The results for all three tools came out as follows:

| ADTool2 | | Previous | | New | |
|---|---|---|---|---|---|
| **Question** | **Average rating** | **Question** | **Average rating** | **Question** | **Average rating** |
| 1 | 1.0 | 1 | 3.66 | 1 | 4.33 |
| 2 | 4.33 | 2 | 1.33 | 2 | 1.33 |
| 3 | 1.33 | 3 | 4.33 | 3 | 5.0 |
| 4 | 3.67 | 4 | 1.0 | 4 | 1.0 |
| 5 | 3.33 | 5 | 3.33 | 5 | 4.66 |
| 6 | 2.0 | 6 | 1.33 | 6 | 1.0 |
| 7 | 1.0 | 7 | 3.66 | 7 | 5.0 |
| 8 | 5.0 | 8 | 1.33 | 8 | 1.0 |
| 9 | 1.33 | 9 | 5.0 | 9 | 5.0 |
| 10 | 5.0 | 10 | 1.0 | 10 | 1.0 |

In the following sections, the given explanation for each question by participants are outlined. For each question, the preferred result (lower/higher is better) is given, along with the question. Each question is responded to on a scale from 1 to 5, which 1 meaning 'totally disagree', and 5 meaning 'totally agree.'

**Question 1: I think that I would like to use this system frequently (higher is better)**
All 3 participants did not enjoy using ADTool2, while all 3 participants for both the previous interface and the new interface quite enjoyed using ADTWebApp. The new interface scored a bit higher, since for the previous interface the non-computer science (CS) participant scored it a 3, while the non-CS participant gave the new interface a 5.

**Question 2: I found the system unnecessarily complex (lower is better)**
Complexity for ADTool2 was very high, being scored a 4 by both the CS participants, and a 5 by the non CS participant. For both versions of the ADTWebApp the complexity is basically as low as it can get, being scored a 2 in both cases by the non-CS participants.

**Question 3: I thought the system was easy to use (higher is better)**
Ease-of-use follows a similar pattern, where ADTool2 got almost the worst score, save for a single CS participant who gave it a 2 rather than a 1, since they figured out the ability to right-click on their own, rather than being told after a while. The previous interface scored very well, but not as high as the perfect score of the new interface. In post-questionnaire discussion, it seemed unanimously agreed that the new interface was simply very intuitive. 'Just clicking and dragging' (translated) is how one of the participants referred to the experience.

**Question 4: I think that I would need the support of a technical person to be able to use this system (lower is better)**
The CS participant who figured out they could right-click in ADTool2 scored it a 2, while the other CS participant scored it a 4, and the non-CS participant scored it a 5. Everyone felt somewhat lost while using ADTool2, but especially when they spent a while trying to reverse engineer how

to use the Term View, before being told that its features can be accessed more simply by right clicking. Both versions of the ADTWebApp received a perfect rating, in both cases all participants were able to find and understand all the features, minus the ADTLang generation button in the new interface, which was not yet implemented when this evaluation was done.

**Question 5: I found the various functions in this system were well integrated (higher is better)**

As far as feature integration, both ADTool2 and the previous interface scored a 3.33, two 3's and a 4. For ADTool2, the reasoning was mostly that there were a lot of features, even if they did not understand them, they seemed complete. For the previous interface, the reasoning leaned more towards everything just feeling well made, but also like it wasn't complete yet. The new interface received an almost perfect rating, except for a single CS participant, due to a few features not being implemented at the time of testing.

**Question 6: I thought there was too much inconsistency in this system (lower is better)**

All 3 interfaces had decent to good consistency, with ADTool2 scoring worst. The reasoning was that although all the features looked similar, there were styling differences between some of the views, which made it feel 'incomplete'. Both ADTWebApp interfaces almost scored perfectly, only the previous interface got a single 2 for inconsistency, because one of the CS participants was confused by where the delete button was, expecting it to be in the Customize tab, rather than the Child tab.

**Question 7: I would imagine that most people would learn to use this system very quickly (higher is better)**

ADTool2 was unanimously regarded as too difficult to learn for most people. Even the participant who found the ability to right-click thought that it was not made clear that was an option, and that most people would simply give up at the sight of code. The previous interface got a good score, being give a 3 and two 4s. The main reasoning being that all participants first spent a while clicking all of the buttons and trying everything out, before getting started. During testing of the new interface, it was observed that all participants very naturally clicked the nodes and immediately knew what to do.

**Question 8: I found the system very cumbersome to use (lower is better)**

ADTool2 was universally seen as cumbersome to use, while both ADTWebApp interfaces were seen as being the opposite. Only a single non-CS participant gave the previous interface a 2, and mentioned that they 'just wanted to type in the box' (translated). But even still, after finding how to edit the contents of the nodes, it was an easy process.

**Question 9: I felt very confident using the system (higher is better)**

All ADTool2 participants left the experience feeling unconfident in their understanding of ADTs altogether. There were no real remarks about this questions from the ADTWebApp participants, basically describing the experience as easy and as 'just clicking stuff'.

**Question 10: I needed to learn a lot of things before I could get going with this system (lower is better)**

This was by far the most clear cut question of them all. All the ADTool2 participants felt that by the end of their test, they still had a lot to learn about how to use this tool. The non-CS participant especially explained that they felt like they barely used the tool. While all the ADTWebApp participants shared the exact opposite opinions. Both interfaces were intuitive enough that no learning was required at all. An example tree and random clicking got all of them to understand the tool.

A final question that was asked of all participants, without measuring a result on the questionnaire, was what their opinion was on ADTs. The CS participants using the ADTWebApp imagined they were useful to people who work in security, even if they had no need for them themselves. While the ADTool2 CS participants had the opposite opinion. One of them simply said that ADTs seem pointless, and the other shared that ADTs entirely were too complex. This is not a proper experiment on how interface design can impact a user's opinion on the technology behind a tool, but it was an interesting observation that poorer perceived interface design resulted in a poorer perception of ADTs among the 6 CS participants. The non-CS participants pretty unilaterally agreed that they didn't really see the point of ADTs, although worded nicer in person.

# 7 Discussion

This research aims to address the following research question: *How can we design intuitive and interactive visualizations for Attack-Defense Trees that make complex cybersecurity concepts accessible to users with varying levels of technical expertise*

When designing the interface, acquiring a solid theoretical framework allowed us to design an interface that achieved all of the requirements that we set out to achieve, as per the results of the evaluation. This framework with clear requirements for the interface helped streamline the design process, allowing us to quickly develop the interface, while avoiding large set-backs due to redesigns. Making the interface itself informative allows users to learn how the tool works on their own during use, rather than having to be taught beforehand. Putting commonly used tools in readily available places ensured that users can use the tool efficiently, while also reducing complexity when learning. All of these concepts put together created a result that is more user-friendly and intuitive than its predecessors.

The new interface complies with rules 1, 2, 5 and 8, shown by the results of the conducted study. Additionally, the interface complies with rule 3, with the presence of the warning system. These rules are derived from Shneierman's book *Designing the user interface* [17].

This research contributes to the field of interface design by highlighting the importance of usability for all users, even for complicated tools. The process for garnering requirements and implementing the interface, as well as the usability principles applied within the interface, serve as examples for other interface designs to build off of. This research allows designers to make their interfaces more intuitive and user-friendly, while also reducing the complexity of development and cost of development through tools like personas rather than frequent real-world testing.

The final interface and the chosen technologies have some limitations. A large limitation of the current implementation is the non-mouse interface. Lots of quality of life features like being able to click on a node, and then tap the backspace key to delete it, would be great additions. Features like adding the ability to select multiple nodes to drag or delete. All of these are somewhat possible, but with large difficulty, as P5.js is not very suitable for them. Frameworks like JointJS might have been preferred for designing interactive UIs, especially when diagram-based. Since JointJS is all vector-based, things like converting the tree into an image would be far simpler to implement than the current work-around in P5.js.

P5.js is also somewhat inefficient at redrawing things on the canvas, seeing as each line, for example, is not an actual line. Once it is created, they are just pixel drawn to the canvas, indiscernible from the others. If you want to delete a line so you can redraw it in a different position or style, it requires the entire canvas to be cleared and all lines to be redrawn, since individual lines cannot be deleted. As such, moving nodes is quite inefficient in general, since the lines connected to the moving node also have to move, but this cannot become much more efficient while using P5.js.

JavaScript and ADTWebApp's backend might also be somewhat limiting for future expansion of the interface. Features such as simulations and calculations of optimal attack paths would have to be calculated server-side, since client-side processing power on a browser is very limited, even if the user's computer is powerful. The current mode of rendering generally would require large changes to make server-side calculations possible.

The results of this thesis are reliable and valid, as seen in the results of the evaluation which show the improvement over other tools and interfaces that serve the same purpose. Although ADTool2 is not a one to one comparison in terms of the capabilities of the tool, the interface is still comparable as at its core it is also a tool to design ADTrees. This purpose is made more user-friendly and intuitive in the new interface than in ADTool2 and the old interface, as made clear by the evaluation. As such, the processes and techniques applied have had clear positive effects on the intuitiveness and usability of the interface.

# 8 Conclusions and Further Research

The new interface seems to be an improvement over the previous interface, although the ratings in the evaluation in Chapter 6.3 were slightly skewed due to ADTool2's ratings generally being vastly different from those of both ADTWebApp interfaces.

The requirements elicited, especially through the use of personas, were very useful in constructing the final interface. Being able to use personas saved a lot of time on regular testing to ensure quality of intermittent interface versions.

The use of mock-ups also helped progress the interface quickly, making it clear quickly when some interface was not going to work. The two sidebar interface sounded very good on paper, but being visually confronted with the much smaller canvas made it clear this was not the way forward. Initially it was chosen to work with the single sidebar, even if the no-sidebar mock-up looked better, but this ended up being a very fortunate decision. The use of requirements from personas made it clear that neither a sidebar nor a full-sized top bar was necessary for the interface. The final interface only uses a simple top bar that utilizes icons, the functionalities of which are not frequently used, which remains easily accessible and takes minimal space away from the canvas.

There is lots of further research possible in regards to both interface design, and this tool in particular. For this tool, examples are discussed in chapter 3. Some requirements found in chapter 5.3 would also require further research, such as designing an interface meant for collaboratively creating ADTs, or improving integration for the tool into other tools. Besides that, creating a desktop app using Electron or Tauri would be of interest, such that users can have a separate application from their browser for designing ADTs.

## References

[1] J. Brooke. Sus: A quick and dirty usability scale. *Usability Eval. Ind.*, 189, 11 1995.

[2] E. Edmonds. *Human Computer Interaction, Art and Experience*, pages 11–23. 02 2014. ISBN 978-3-319-04509-2. doi: 10.1007/978-3-319-04510-8_2.

[3] D. Fallman. Design-oriented human-computer interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '03, page 225–232, New York, NY, USA, 2003. Association for Computing Machinery. ISBN 1581136307. doi: 10.1145/642611.642652. URL https://doi.org/10.1145/642611.642652.

[4] B. Ferreira, G. Santos, and T. Conte. Identifying possible requirements using personas - a qualitative study. In *Proceedings of the 19th International Conference on Enterprise Information Systems - Volume 2: ICEIS,*, pages 64–75. INSTICC, SciTePress, 2017. ISBN 978-989-758-248-6. doi: 10.5220/0006311600640075.

[5] M. Fraile, M. Ford, O. Gadyatskaya, R. Kumar, M. Stoelinga, and R. Trujillo-Rasua. Using attack-defense trees to analyze threats and countermeasures in an atm: A case study. In J. Horkoff, M. A. Jeusfeld, and A. Persson, editors, *The Practice of Enterprise Modeling*, pages 326–334, Cham, 2016. Springer International Publishing. ISBN 978-3-319-48393-1.

[6] O. Gadyatskaya, R. Jhawar, P. Kordy, K. Lounis, S. Mauw, and R. Trujillo-Rasua. Attack trees for practical security assessment: Ranking of attack scenarios with adtool 2.0. In G. Agha and B. Van Houdt, editors, *Quantitative Evaluation of Systems*, pages 159–162, Cham, 2016. Springer International Publishing.

[7] R. R. Hansen, K. G. Larsen, A. Legay, P. G. Jensen, and D. B. Poulsen. Adtlang: a programming language approach to attack defense trees. *International Journal on Software Tools for Technology Transfer*, 23(1):89–104, Feb 2021. ISSN 1433-2787. doi: 10.1007/s10009-020-00593-w. URL https://doi.org/10.1007/s10009-020-00593-w.

[8] B. Kordy, P. Kordy, S. Mauw, and P. Schweitzer. Adtool: Security analysis with attack–defense trees. In K. Joshi, M. Siegle, M. Stoelinga, and P. R. D'Argenio, editors, *Quantitative Evaluation of Systems*, pages 173–176, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[9] B. Kordy, S. Mauw, S. Radomirović, and P. Schweitzer. Attack-defense trees. *Journal of Logic and Computation*, 24, 02 2014. doi: 10.1093/logcom/exs029.

[10] V. Lenarduzzi and D. Taibi. Mvp explained: A systematic mapping study on the definitions of minimal viable product. In *2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 112–119, 2016. doi: 10.1109/SEAA.2016.56.

[11] J. R. Lewis. Usability: Lessons learned ... and yet to be learned. *International Journal of Human–Computer Interaction*, 30(9):663–684, 2014. doi: 10.1080/10447318.2014.930311. URL https://doi.org/10.1080/10447318.2014.930311.

[12] J. Lyly-Yrjänäinen, L. Aarikka-Stenroos, and T. Laine. Mock-ups as a tool for assessing customer value early in the development process. *Measuring Business Excellence*, 23(1):15–23, Jan 2019. ISSN 1368-3047. doi: 10.1108/MBE-11-2018-0096. URL https://doi.org/10.1108/MBE-11-2018-0096.

[13] L. L. McCarthy. p5.element. 2013. URL https://p5js.org/reference/p5/p5.Element/.

[14] L. L. McCarthy. position(). 2013. URL https://p5js.org/reference/p5.Element/position/.

[15] M. Mohalaia. Implementing a user interface for attack-defense tree webapp using human-computer interaction principles. 2023. URL https://theses.liacs.nl/2644.

[16] S. Paul. Towards automating the construction & maintenance of attack trees: a feasibility study. In B. Kordy, S. Mauw, and W. Pieters, editors, *Proceedings First International Workshop on Graphical Models for Security, GraMSec 2014, Grenoble, France, April 12, 2014*, volume 148 of *EPTCS*, pages 31–46, 2014. doi: 10.4204/EPTCS.148.3. URL https://doi.org/10.4204/EPTCS.148.3.

[17] B. Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley Longman Publishing Co., Inc., USA, 3rd edition, 1997. ISBN 0201694972.

[18] B. Shneier. Attack trees. *Dr. Dobb's Journal*, 1999.

[19] B. Shneier. *Secrets and Lies: Digital Security in a Networked World*. Wiley, 2015.

[20] S. Tamjidi and A. Shameli-Sendi. Intelligence in security countermeasures selection. *Journal of Computer Virology and Hacking Techniques*, 19:1–12, 07 2022. doi: 10.1007/s11416-022-00439-w.

[21] P. Wang and J.-C. Liu. Improvements of attack-defense trees for threat analysis. In J.-S. Pan, C.-N. Yang, and C.-C. Lin, editors, *Advances in Intelligent Systems and Applications - Volume 2*. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-35473-1.

[22] B. Xu, Z. Zhong, and G. He. A minimum defense cost calculation method for attack defense trees. *Security and Communication Networks*, 2020. ISSN 1939-0114. doi: 10.1155/2020/8870734. URL https://doi.org/10.1155/2020/8870734.