Threat Modeling Tool

Requirement Report

Author: Aayush Mailarpwar, Cooper Zuranski, Kristopher Bright, Joe Swiney

Sponsor: Luke Thomas

TABLE OF CONTENTS

LIST OF FIGURES

BACKGROUND

Threat modeling is used to analyze, discover, and describe possible security vulnerabilities of a system. The applications are much broader than software, though. Threat modeling can be used for security systems, aircraft, automobiles, embedded controls, phones, cameras, wireless equipment, and even things as simple as physical rooms. Though most of these things have software involved, that is not the point. A weak hinge is a security vulnerability to a room.

This application doesn't serve to evaluate numerical probabilities or analyze anything mathematically directly. This is in the same way that UML diagrams are not how you actually create software. They merely represent the concepts as a discussion point. However, these analogies are separate in the fact that threat modeling needs to be visible and comprehendible to more than just those who work on security.

Threat modeling describes what is being modeled in more context than just security first. This is part of the sharable aspect. Everyone involved must understand the constraints in order to contribute or understand the threats themselves.

The most important detail about threat modeling is that it is intended to be a living document. It should be structured in such a way that it can be edited and expanded upon. This is especially true when it comes to details. First, the document will be very abstract. This is okay, though, as the intent is to get people involved in the product design to properly think about all the possible threats to the system.

The goal from here is to create awareness about the possible threats. Then these can be modified as time progresses, allowing for a dynamic look at security. Assumptions will have the opportunity to become validated into actual potential threats, or simply thrown out.

Although the goal of threat modeling mostly pertains around identification, some planning on how to mitigate these threats must also be included. This obviously helps the project move forward, but is important from the threat modeling perspective, as many times a solution to one threat may be a threat within itself.

These models should help the user understand where vulnerabilities are, which are most relevant, how to prioritize solving them, and what constitutes "enough."

MOTIVATION

In recent history, everything we interact with in a day has become more interconnected than ever before. Mobile phones were a large first step, with the shifting of communication methods, many more threats were introduced into everyday life. However, this only continued to evolve as technological capabilities and demands evolved faster than their security. Now we are to the point that we have watches with ECGs built into them, home assistants with microphones constantly listening, and cars that are told how to drive better with new software updates.

In many cases, security has been an afterthought. However, not all of this is driven by supply and demand of new technologies. Most of it is actually a result of how security is considered in engineering and design. According to the International Council on Systems Engineering, the vast majority of cyber tools and techniques are applied very late on in the design process. This is a major roadblock as by this point, many things may be unable to be changed, and if so, are incredibly more expensive to do later on.

Part of the reason this occurs is because security experts are few and far between. Education on this topic was not even a standard standalone item until recent years. Having to involve what few "experts" exist at a given company all the way from the design stages may not be possible, let alone cost-effective.

Herein lies the motivation for our threat modeling tool: a way to not only share a collective knowledge about systems security, but actively teach users about how to identify and assess these sorts of security risks. They almost certainly will not know the best solution to all of these problems, but it sets a precedent for prioritizing and talking about security in every step of the design process. This leads to productive conversations with people who are experts, leading to more secure solutions without full involvement.

We seek to create a program to do threat modeling from the perspective of someone who knows absolutely nothing beforehand and can learn as a result of the program. This program will not help them identify all of the vulnerabilities in their system, but it will help the learn to think about, recognize, and address them.

TECHNICAL REQUIREMENTS

This section consists of an overview of the basic requirements for the Threat Modelling Software that the sponsor has provided to us. It provides a tentative list of requirements in multiple areas including:

User Interface

This shall be the point of human and computer interaction and basic communication between what the user wants/needs related to what the system has to offer. Our goal is to make an extremely simple UI that any individual without software experience can use. It shall be composed of an interactive WYSIWYG (What You See Is What You Get) GUI (Graphical User Interface) that is situated in a web browser.

Assisted Functions

This shall include help units including ToolTips for any kind of guidance that would facilitate a new user. Visual guidance will also be provided to illustrate STRIDE-based threats that have been considered on each model object. VG will have the option to turn on and off if existing users are familiar with the working of the system. Layout and color schemes will be used in order to be readable and usable for persons who are colorblind.

Import / Export Functionality

One of the important functionalities shall be the ability of the Threat modelling tool to be able to be connect to various mechanism feeding in complex predictive data into an encapsulated version that is able to be exported into easily readable files including .docx (Word), .pdf and so on. This will essentially be not only human readable but also no software development skills will be required to perceive the threat information since all data will be stored in YAML format.

Save and Load

The user shall be able to save the model and data at instance of the session and this will be loaded in a future session exactly how it was left off.

Version Control

From start to finish, our project will be regularly updated using GitHub which is an industry collaborative version control. This will be accessible only by developers and not users.

STANDARDS

As this is open-source software, there are not many standards by which we are constrained. However, we will choose to adhere, at a minimum, to a standard of accessibility with regard to conditions such as colorblindness.

From a design perspective we must prioritize extendible code over designs which may make more contextual sense. If design is not modular, the open-source aspect is ignored as much effort will be required to change or add anything. This project is intended to have a long lifespan and must be treated as such.

Likewise, code must be maintainable. This project will change hands over the years, and code must be human readable and well described. There is no place for secretive or elaborative code within this project. With the nature of what we are doing, speed is not a primary concern, as there is no complex computing in the backend.

There will be a formal coding standard defined later, and this document will be updated to reflect it. Since we will be using the JetBrains IDE we will have access to ReSharper which we will use as a code linter. There are already existing linters for the languages we will be using.

Code reviews by a member of the team who did not write the code will be required before a merge. Sponsor review will be required before a release.

CONSTRAINTS

There shall be:

- No admin permissions required to use
- Source control
- Version control
- The capability to run on an average laptop
- Metadata about model elements
- Unix style time recording
- No need for training to use the software
- Colorblind adaptability
- Export, save, and load abilities
- Human readable data files in YAML or similar format
- Testable code
- Open-source licensing
- Toggleable visual features

There should be:

- Local runtime in a browser without requiring remote hosting
- Backend written in Rust
- Frontend written in JavaScript

SAFETY/ OTHER REQUIREMENTS

At the time, there are no known safety requirements. Our product is intended to run locally, so there is no concern over network security specific to the software. The product doesn't identify or evaluate threat. It only models it, so there is no domain specific responsibility. We must ensure that the information input into the tool is saved accurately and that access to the data is reliable. Failure to meet this requirement will result in corrupted user data and render the software useless.

SYSTEM/ SUBSYSTEM BLOCK DIAGRAMS

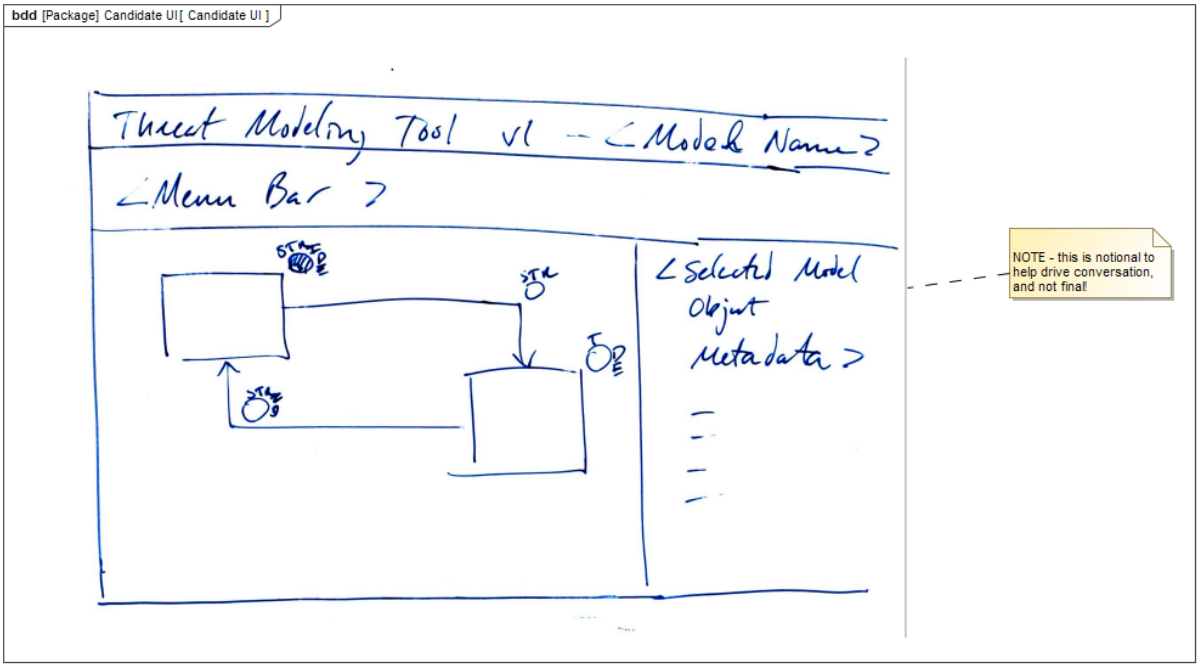

Figure 1: Candidate User Interface

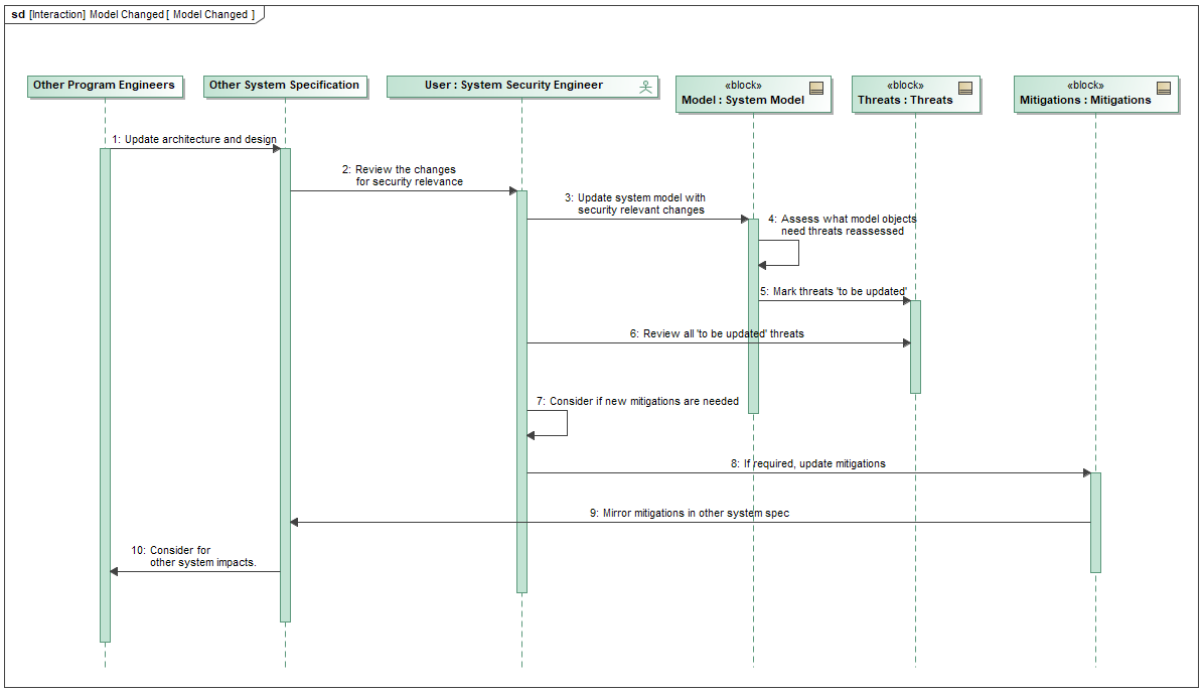

Figure 2: Process for changing a model component

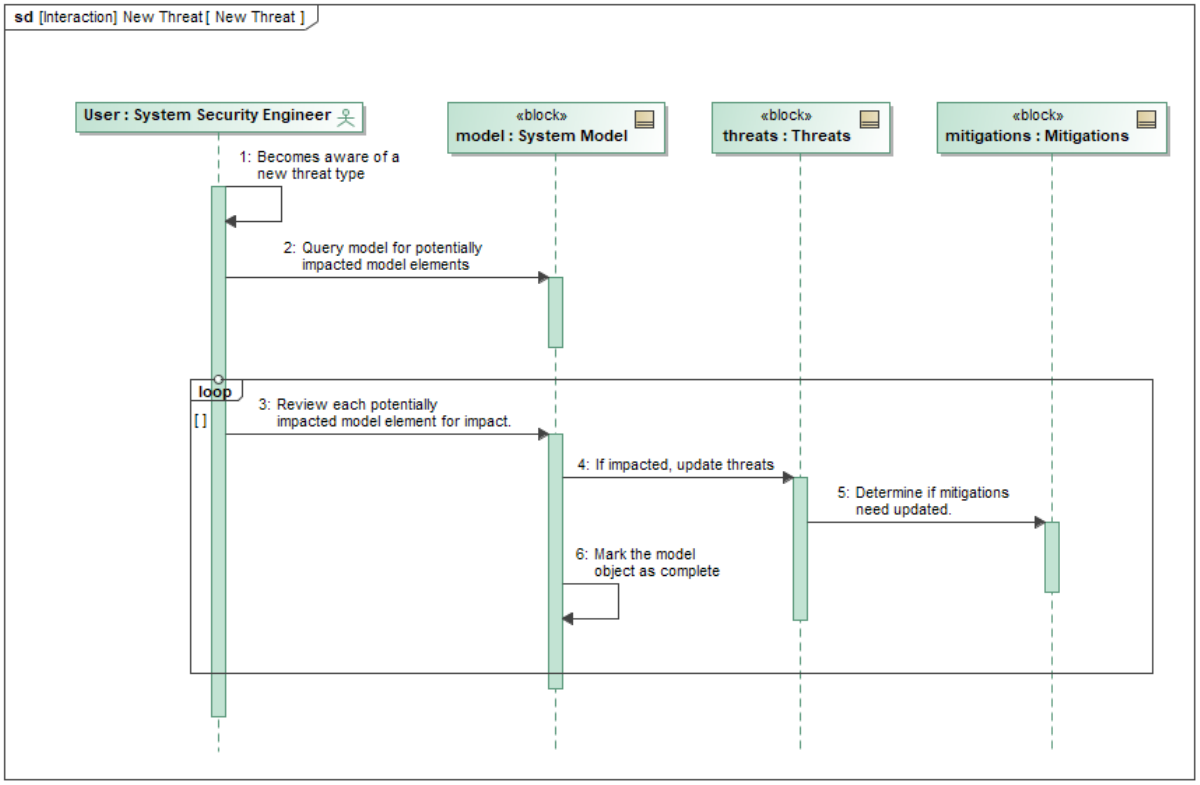

Figure 3: Process for adding a new threat

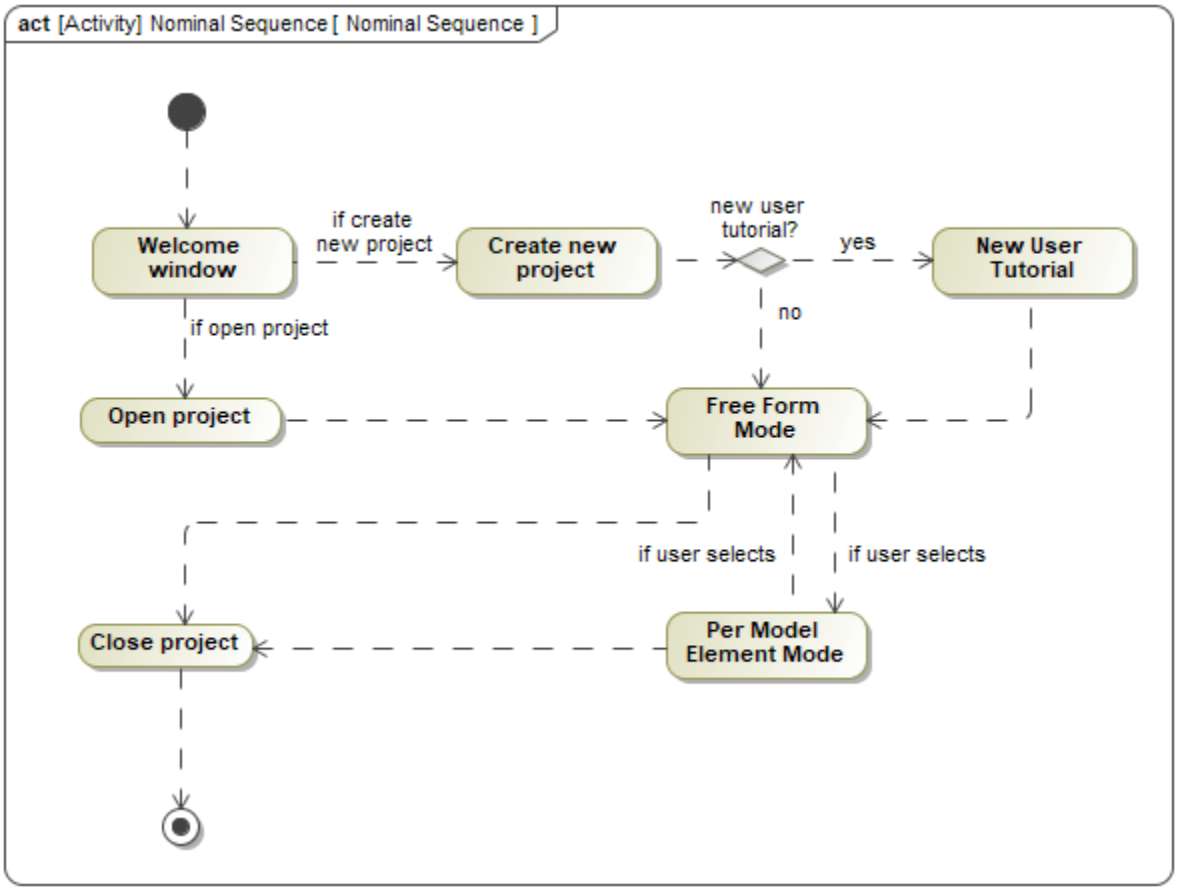

Figure 4: Nominal sequence of user interaction

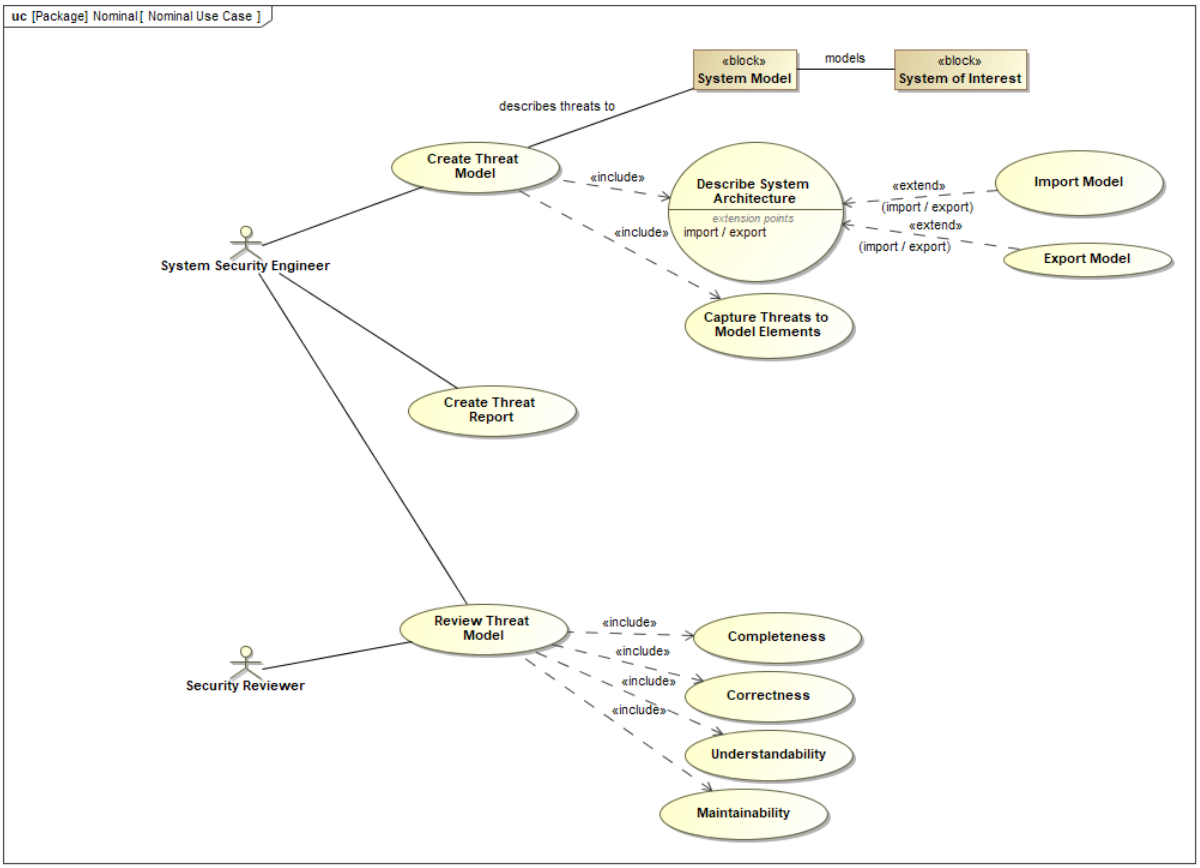

Figure 5: Nominal use cases

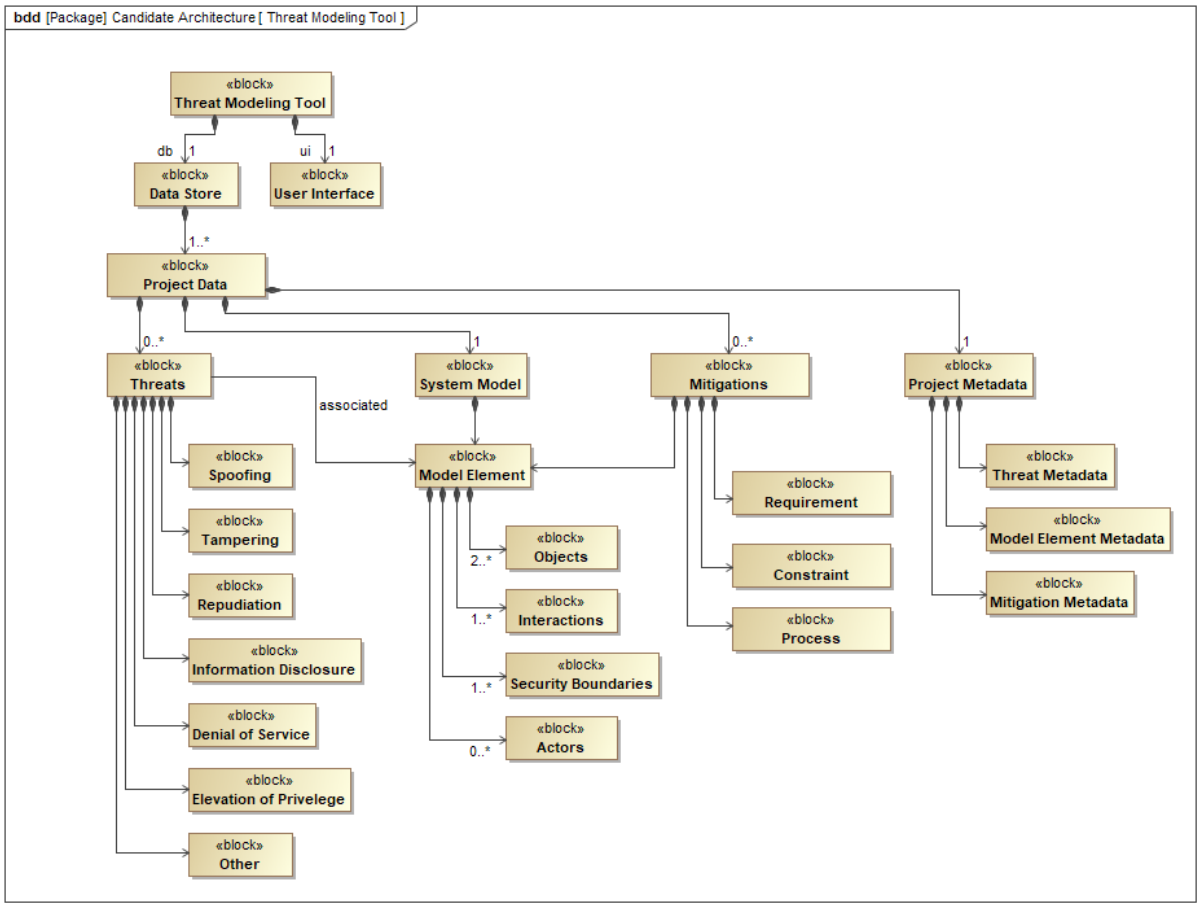

Figure 6: Block diagram of threat modeling program architecture

TEST PLAN

This is one of the vital phases of our project that will be a large span of time where our team tries to break and validate the software. This will not only be towards the completion phase of the project but will have elements that must be done during each accomplishment of the program. The overall software testing procedure comprises of Unit testing, Integration testing and Systems testing.

Unit Testing

This is the first building block of software testing and is exactly what it sounds like, testing each unit of the software. Small modules of code that have a specific task will be tested repeatedly and reports will be created on each iteration to document what went wrong and how to fix it. These testing units could go from checking for proper input from threat modelling tools to how a chunk of code looks in UI. This will be manually created but automated in testing whether the code is successful in nature or not.

Integration Testing

This is a step up from Unit Testing and includes testing how different units and modules of the software interact. This is less intense than unit testing due to an additional level of abstraction where we are dealing with the multiple outputs that those units spit out and seeing how they interact with other units. Since data will be shared by different modules, it is very important for Integration testing to identify possible leaks or failures.

Systems Testing

Automated Testing: The system and rudimentary functionality of identifying threats and running through multiple phases of the architecture that is displayed on GUI will ran with an automated test. This will be an approximate of 95% or more of the code and functions that can be assigned to a test harness.

Manual Testing: There will exist some elements in the test harness that will fail to recognize faults including elements of incorrect display, unclear data and so on that will need a manual procedure of testing.

POTENTIAL PROBLEMS AND SOLUTIONS

There ought to be multiple problems and challenges that the team will encounter during the phases of this project. The team is to work on making sure the requirements and expectations are understood, designing the software, developing the software, testing, and integrating the built software, and finally deployment of the tool goes smoothly. The potential bumps are as follows:

Learning Curve

Our team is to understand the roots of Threat Modelling which is an area that all members have minimal experience of. Although our team is well versed with designing software and the principles linked to them, there exists a gradual process of familiarizing ourselves with languages and frameworks – Rust, YAML, JavaScript, etc. A solution to this is starting out early and making sure that all concepts are cleared before advancing to next steps.

Development

Since there is no background work done on this project, there will be designs principles we will conform to. This will be a challenge since we must figure out many parts and know that they may change during the course of the project. Another solution is coordination and multiple code reviews before pushing or merging code.

GUI Research

One of the requirements are to make the GUI extremely easy to understand, in the sense that someone with no training should be able to navigate through the system flawlessly. Tooltip usage must be a minimal and therefore a solution would be to have a member to work with researching about intuitive learning through simply GUI characteristics.

Testing Environment

This involves the logistics around the multiple testing phases and how we will go about each phase – automated and manual testing. This will be a challenge since creating testing environments is difficult and figuring out edge cases is troublesome. A solution is appending a list of possible bugs and working close with sponsors at times of testing due to his expertise.

PROJECT PLAN

Determine Requirements	Design Initial Iteration	Threat Modeling Training	Modify Initial Iteration Design	Initial Iteration Implementation	Write Tests	Review with Sponsor	TBD based on Feedback
Weeks 1-5							
	Weeks 5-7						
		Week 8					
			Weeks 9-11				
				Weeks 11-16			
					Weeks 17-20		
						Weeks 21-22	
							Ongoing

Figure 7: Gantt Chart of tentative project timeline

This plan is tentative. We still must undergo training on Threat modeling before even being able to properly design anything. We can then decide on design strategies and create UMLs to describe them. After this we can figure out how we will go about development by setting up source control, IDEs, and linters. After feedback, this plan will likely be modified. We will start with the framework of the application to make sure back and front are connected how they should be and define abstract behavior. We will hopefully create more specific implementation progress into next year along with some base tests. Then it will be a regular development cycle of adding features and reviewing to ensure fulfillment of the sponsor's desires.