Dynamic Duo

Security Protocol Tool for Security Classes

By: Owen Forsyth and Daniel Mead

Table of Contents

Table of Contents: 2
Introduction: 3
Executive Summary/Overview: 3
Functional Requirements: 4
Non-Functional Requirements: 5
Potential Risks: 6
Timeline: 6
Glossary: 7

Introduction

Understanding cybersecurity concepts can be challenging, especially when expressed in formal cryptographic notation. While theoretical discussion may seem straightforward, applying ideas in practice often proves difficult for students. Programmers also recognize the complexity of implementing secure networked code. Visual aids like diagrams can make these abstract ideas more accessible by providing a clear representation of protocols and their interactions.

Our client, Professor Zach Kissel, envisions a program that can be utilized as a teaching aid for his courses like cryptography and network security. The purpose of this project is to develop a **user friendly application** that supports students learning about cryptographic protocols.

The system will provide an environment where users can describe protocols using a simplified syntax, visualize them as diagrams, and analyze their security properties. Finally, this tool will emphasize ease of use and accessibility, requiring minimal setup so that students can focus on learning rather than installation and configuration.

Executive Summary/Overview

The System that we create will help students and Professor Kissel in Security classes. The system will be designed as a user-friendly menu to understand message passing and what happens in different configurations. The user will start by entering in a series of passed messages. These messages can be inputted manually or imported from a file. The system will then return a SVG showing what the message passing looks like as an image. It will also return the starter java code to write the protocol and an analysis of the message passing.

This will help students visualize the message passing, gain an understanding of how to write the code, and learn what series of messages are secure. This will assist Professor Kissel since he will not have to hand draw out graphs of these

protocols. He can also require students to submit these images in assignments to show that they have an understanding of the protocols.

Functional Requirements

- Protocol Input
    - The program shall provide an editor window where the user can enter multiple lines of protocol description, using a defined ASCII based syntax.
    - The program shall allow the user to load a protocol description from a file into the editor
    - The program shall allow the user to save a protocol description and display error messages when invalid syntax is detected.
- Diagram Generation
    - The program shall provide a button to the user, for the system to automatically generate a protocol diagram based on the user's inputted protocol description
    - The program shall allow the user to export the generated diagram as an SVG file
- Adversary Knowledge Tracking
    - The program shall analyze the protocol execution to determine what data each entity, including an adversary, can observe.
    - The program shall infer additional knowledge the adversary can derive from observed messages.
    - The program shall flag cases where a secret can be fully recovered by the adversary, and highlight each step that allows the adversary to learn information about the secret.
- Code Generation
    - The program shall generate starter Java code that implements the described protocol
    - The program shall allow the user to save the generated Java code to a file.

    - The generated code shall depend on the merrimackutil and Bouncy Castle libraries.
- User Interface Features
    - The program shall provide menu based navigation for all major operations (file input/output, diagram generation, analysis, code generation).
    - The program shall provide adjustable font sizes, support for both light and dark modes, and customizable color palettes.
    - The program shall provide in-application documentation or help features accessible from the user interface.

Non-Functional Requirements

Hardware Constraints

- The system must be a standalone application
- This application must be downloadable onto Windows (Windows 10 (64-bit)), Mac (macOS 10.14), and Linux(Ubuntu 18.04+, Debian 10+, Fedora 29+, CentOS 7+) machines
- The application will not be hosted on the internet

Usability

- The system must have a user friendly interface that minimizes the number of clicks to perform saving files, uploading files or executing code
- The system must allow for files to be easily uploaded and saved from the interface

Reliability

- The system only needs the user machine to have the processing power to run

Performance

- The system only needs to run one instance of message passing at a time since no user can load in multiple message passings at the same time

Supportability

- The system will need to be updated with new protocols if more are wished to be added
- The system may need to be adjusted as Java is updated

Potential Risks

The potential risks for the user are if you exit out of the system without saving your work it can be lost.

Timeline

Sprint 1(9/4 - 9/18)
- Requirements Document Draft (9/16)

Sprint 2(9/18-10/2)
- Requirements Document Final (9/23)
- Create Base UI Design

Sprint 3(10/2-10/16)
- Functional Specifications Draft (10/7)
- Functional Specifications Final (10/16)
- Create Input

Sprint 4(10/16-10/30)
- Create SVG
- Create Starter Java Code

Sprint 5(10/30-11/13)
- Create Message Analysis
- Project Testing

Sprint 6(11/13-12/2)
- Poster Draft (11/25)
- User Manual Draft (12/2)
- Package the project

Sprint 7(12/2-12/9)
- Poster Final(12/4)
- User Manual Final (12/9)

- Technical Manual Draft (12/9)

Glossary

- SVG (Scalable Vector Graphics): an XML-based vector image format for two-dimensional graphics that can be scaled to any size without losing quality or detail
- IDE: Integrated Development Environment, software that combines all the needed tools for a programmer, such as writing, compiling, and running code.
- Protocol: A protocol is way of encrypting a message and passing it to someone else that keeps the contents of the message and keeps the message secure from third parties