Rust Cleaners

Solana Code Analysis as a Service(SCAaaS)

Arsh Kabarwal, Joseph Carpman, Kipp Corman, Olivia Ornelas

Contents

- Problem and Goal
- Design
- Alternative Products
- Design Analysis
- Project Management

Problem Background

- Decentralized apps and smart contracts on the blockchain show exceptional promise for increased transparency and easier deployment of software-as-a-service products
- Most current code review tools are not made with blockchain deployment in mind. On the blockchain, it can be difficult or impossible to patch code that has been published, so it is important to find bugs and exploits before the code is published.

Needs Statement

- The current barrier to entry of blockchain smart contract development is too high. The difficulty of modifying or updating deployed smart contracts demands robust code review tools to catch programming errors before deployment.
- Currently, code review tools that analyze code meant for blockchain requires the user to download the tool. Additionally, most tools require the user to send portions of their code directly to the company in order to get a quote for how much the tool will cost. Our project aims to create a low cost tool that doesn’t require the user to download anything in the form of Software-as-a-Service.

Goal

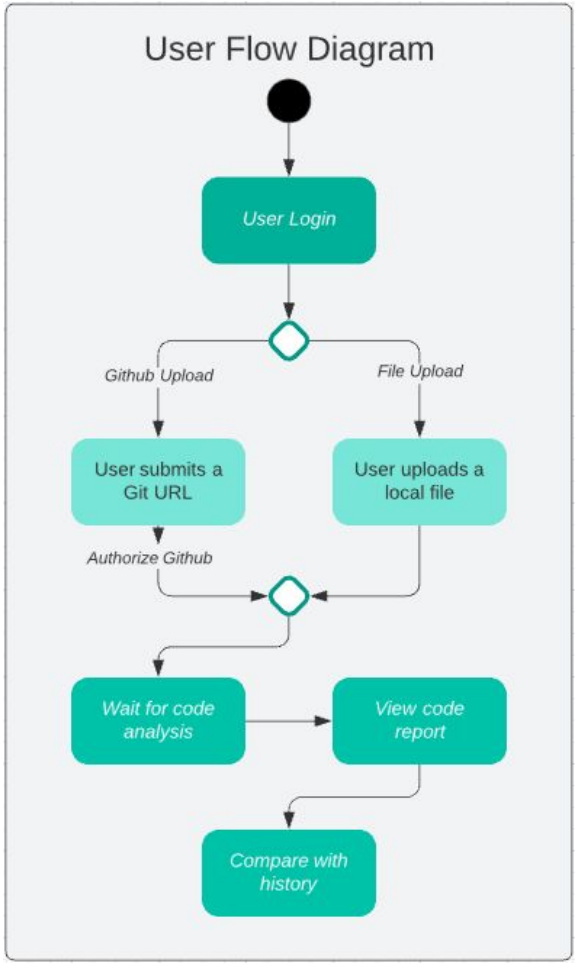
The goal of this project is to implement a Software-as-a-Service tool which allows users to quickly and clearly review their code before it is deployed to the blockchain. In addition to this, the product will review a user’s code using multiple tools and present the results in an easy to understand way.

Objectives

- Identify at least as many vulnerabilities in their code as the existing products do.
- The product will be able to:
  - Review code using at least 3 code review tools and produce results from each on the same dashboard.
  - Accept code from at least file upload and github url.
  - Preserve up to 10 prior submissions for comparison.
  - Preserve code review report data for historical modeling.

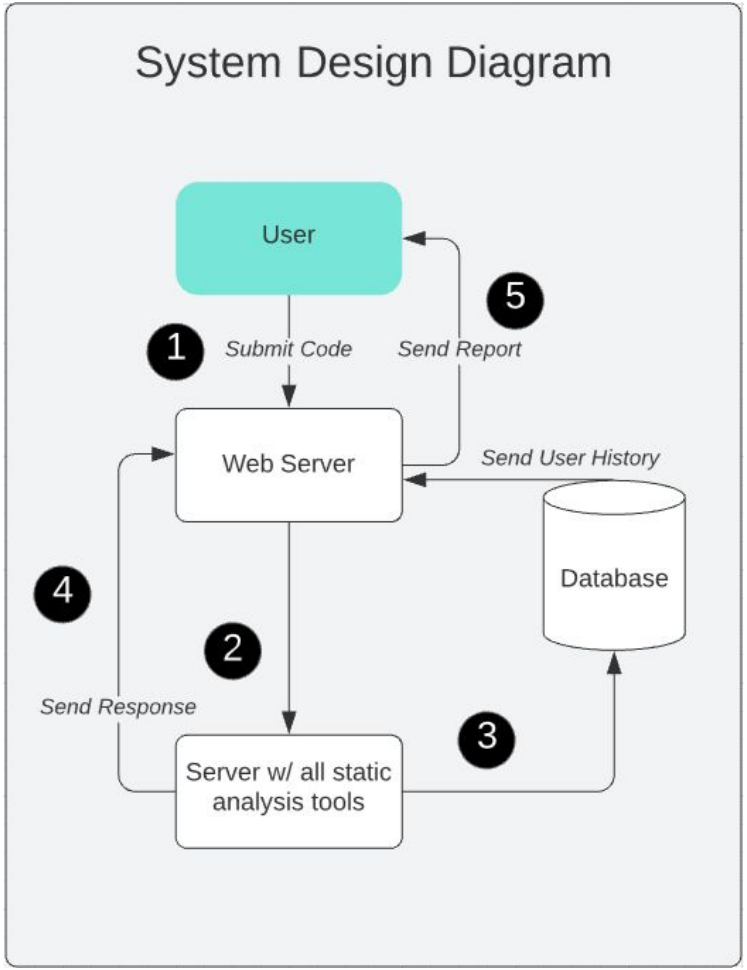
Proposed User Flow

1. User logs in.
2. User either uploads a local code file or provides a Git repository.
3. User waits a short time for the code analysis.
4. User can view the new code report.
5. User can review and compare old reports to find trends.



Proposed System Design

1. User submits code.
2. Web server delivers code to static analysis server.
3. Static analysis server sends response to the database and web server.
4. Web server creates the report using historical data and the response.
5. User views the new report.



Alternative Products

- **Bloqchain Science Code Review:** Does not support Rust (our focus), and requires users to contact the developers in order to use the tool. It is geared towards businesses.
- **Soteria:** Specifically for Solana just like our project, but it takes 24-48 hours to get a response after submitting code. Does not support Github auto analysis.
- **Anchain.AI:** Requires users to request a demo in order to use the service, and is again geared towards businesses or very large projects.
- **SonarQube:** Does not support Rust, and requires users to download an application rather than submitting through the website.
- **DeepSource:** Most similar product to ours, supports Github auto analysis but doesn't support direct code uploading. Uses their own static analysis tool rather than pulling together many open-source tools like we plan to.

Design Validation

- Results in comparison to competing solutions.
- Front-end user friendliness
- Review and report response time
  - With large code bases
  - With multiple users uploading at once
- Historical data recall
  - Fast queries
  - Remembers enough of the past to be useful
- Database security
  - Are usernames and passwords stored properly?
  - Is code history confidential?
- Git auto-report on push

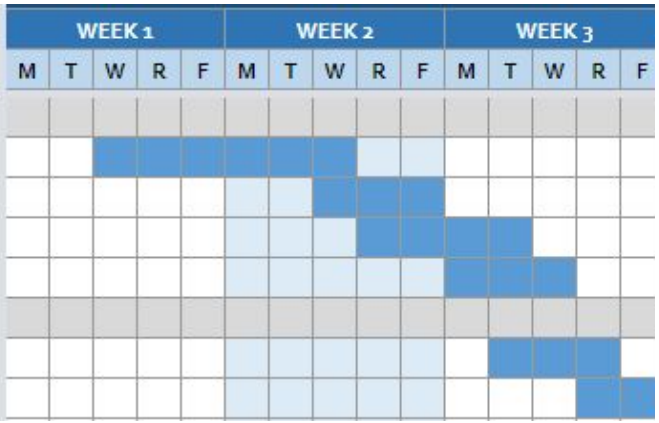
Economic Analysis / Budget

- Marketable: Market has many competitors, most are not so easy to use or rely on their own analysis tool. Cost for building the project is mostly just servers.
- Sustainable: Open source code analysis tools can easily be swapped if needed, treated as black box.
- Manufacturability: Even in the worst case, should be faster than most competitors. Follows all regulations.

Task Schedule - Phase 1

The first phase of the project is focussed on getting each part of the project working locally and individually before combining them. During this phase, basic features will be drafted such as account management.

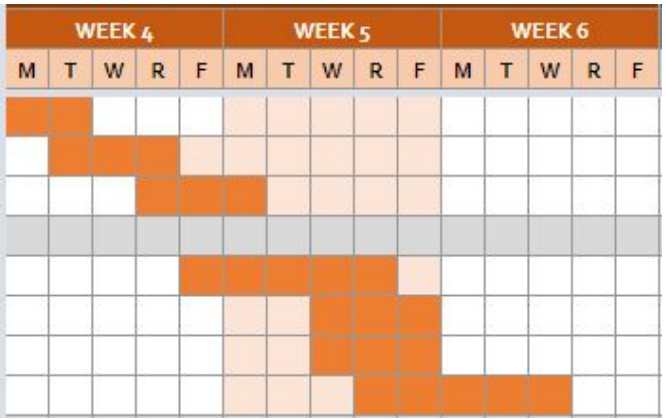
WBS NUMBER	TASK TITLE	TASK OWNER	START DATE	DUE DATE	DURATION (DAYS)
1	User Interface				
1.1	Basic code upload		2/16/22	2/23/22	7
1.2	Account login/logout/creation		2/23/22	2/25/22	2
1.3	View code vulnerabilities		2/24/22	3/1/22	7
1.4	View analysis report		2/28/22	3/2/22	2
2	Pipeline				
2.1	Host Website		2/28/22	3/3/22	3
2.2	Set up server for code analysis		3/3/22	3/8/22	5



Task Schedule - Phase 2

The second of the phase of the project has to do with setting up the web server to automatically accept code and return the reports. The team will start with no code analysis to keep the pipeline simple. Once the pipeline works, the blackbox code analysis tools will be added to the pipeline.

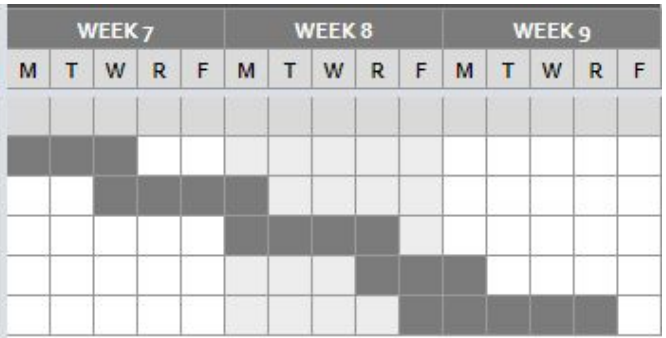
WBS NUMBER	TASK TITLE	TASK OWNER	START DATE	DUE DATE	DURATION (DAYS)
2.2	Set up server for code analysis		3/3/22	3/8/22	5
2.3	Send uploaded code to server		3/8/2022	3/10/2022	2
2.4	Send test report back to the web server		3/10/2022	3/11/2022	1
3	Code analysis				
3.1	Run cargo-audit on code		3/11/22	3/16/22	5
3.2	Run Soteria (free version) on code		3/14/22	3/17/22	3
3.3	Run an additional code review tool		3/14/22	3/17/22	3
3.4	Polish MVP		3/17/22	3/23/22	6



Task Schedule - Phase 3

The last phase of the project is focussed on adding the project's more unique features and working on stretch goals. The main goals of this phase will be the multi-tool report consensus, code history insights, and github auto-report features.

WBS NUMBER	TASK TITLE	TASK OWNER	START DATE	DUE DATE	DURATION (DAYS)
4	Additional Features				
4.1	Github upload		3/28/22	3/30/22	2
4.2	Report consensus		3/30/22	4/4/22	4
4.3	Auto-report and email		4/4/22	4/7/22	3
4.4	Save report and code history		4/7/22	4/11/22	4
4.5	Display historical data with diffs		4/8/22	4/14/22	6



Project Management

- Team leader will work on the systems design
- Team member will facilitate the distribution of tasks depending on everyone’s strong suits
- Continual progress checks at weekly meetings
- We will use Gantt chart and weekly reports, to ensure that the development is on schedule
- Meet on Mondays and Wednesdays
- All finance decisions will be made by the whole team

Teamwork

Arsh: Project lead
Work on the system design
Handle research on blockchain and Solana

Kipp: Work on the systems and hardware
Handle the communications between the local server and the Amazon Web Services
Implement the static analysis tools on the local server

Joseph: Create user login and user code upload functionalities
Communicate with professor and TA about project details and questions
Choose the various static analysis tools that we will be using

Olivia: Creating the front end design
Integrating the GitHub auto update functionality
Establishing the requirements for the user application

Societal, Safety and Environmental Analysis

- Society Analysis:
  - Beneficial impact because the service lowers the barrier to entry of blockchain development.
- Safety Analysis:
  - Possible loss of confidentiality through because the code history requires user’s code to be stored on our database.
- Environmental Analysis:
  - Product uses cloud computing services, which can be more intensive on the environment depending on utilization.

Any Questions?