

Rust Cleaners

Solana Code Analysis as a Service(SCAaaS)



Arsh Kabarwal, Joseph Carpman, Kipp Corman, Olivia Ornelas

Contents

- Problem and Requirements
- Design Alternatives
- System Description and Design
- Project Scheduling
- Test Plan and Demonstration
- Team Management
- Project Concerns

Problem Background

- 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.
- 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

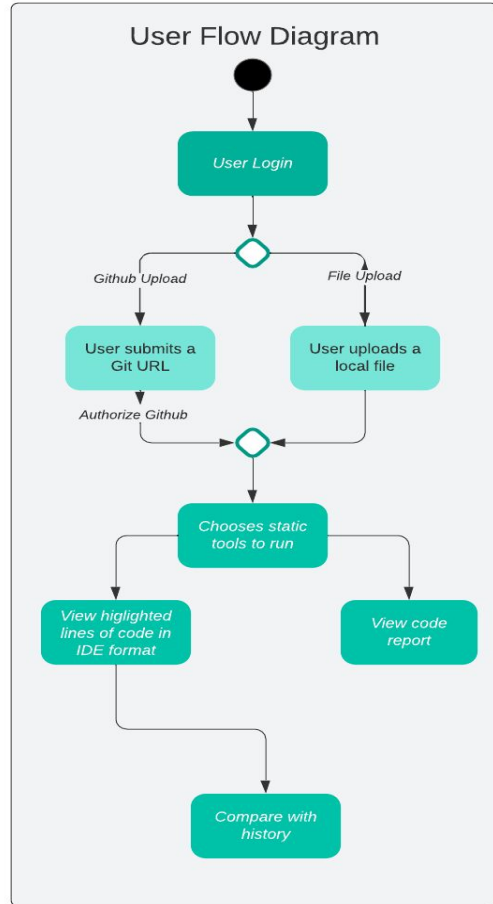
The goal of this project is to implement a Software-as-a-Service tool which allows users to quickly and clearly review their code before it is deployed to the blockchain. In addition to this, the product will review a user's code using multiple tools and present the results in an easy to understand way.

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 pages.
 - 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.

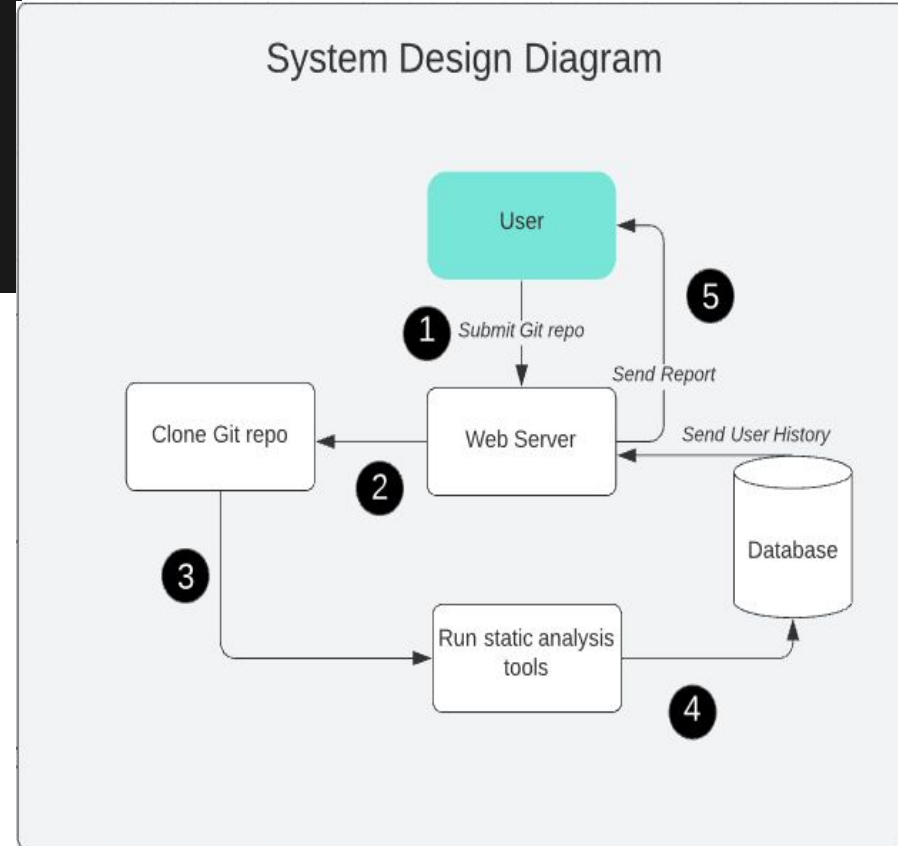
Proposed User Flow

1. User logs in.
2. User either uploads a local code file or provides a Git repository.
3. User chooses which tool to run
4. User can view the results from the tools
5. User can view the code report.
6. User can review and compare old reports to find trends.



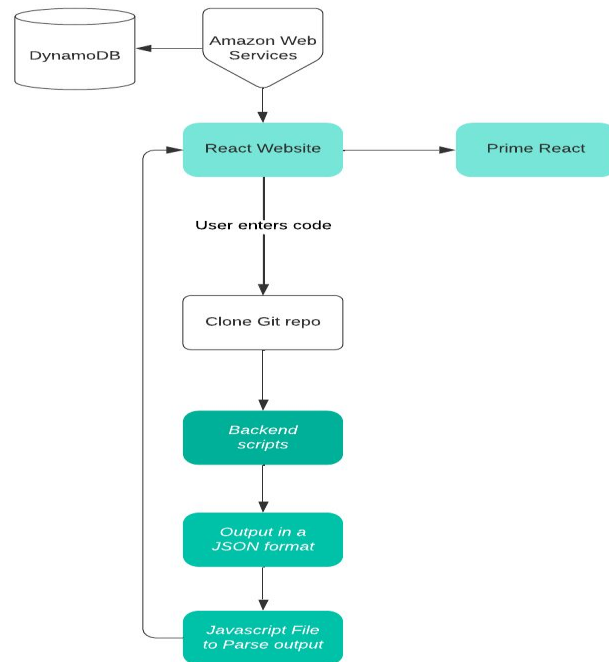
Proposed System Design

1. User submits code to web server
2. Code is cloned and downloaded to local system
3. Tools are run on the local download
4. Web server creates the report using historical data and the response.
5. User views the results



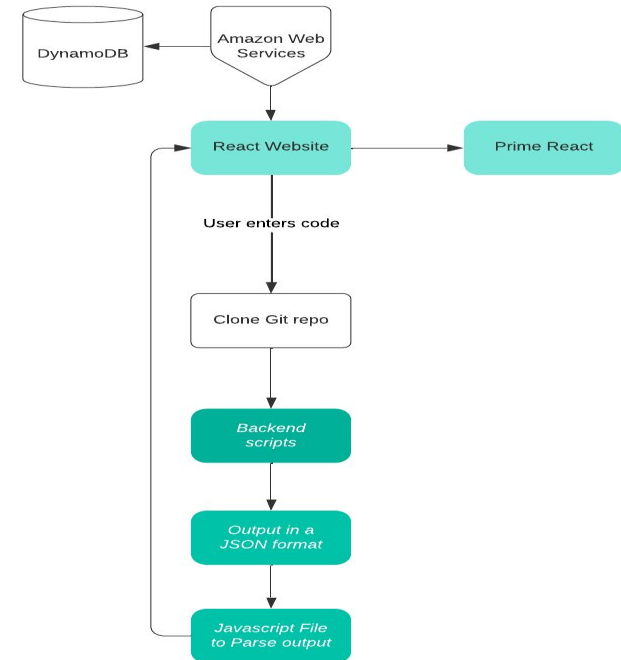
Subsystem Design

1. User submits code (Git URL)
2. Repo is cloned on the web server
 - Use GitHub API to determine branch
3. Check database to see if repo is already there
 - If yes, can show diff or use cache
4. Bash scripts runs the tools
 - Search subdirectories for Rust files, .lock files
5. Recreate file directory as a .json file

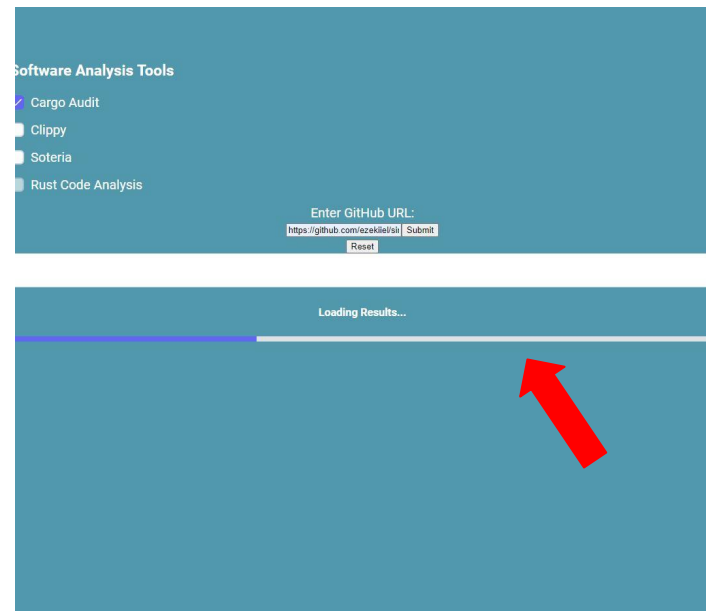


Subsystem Design Cont.

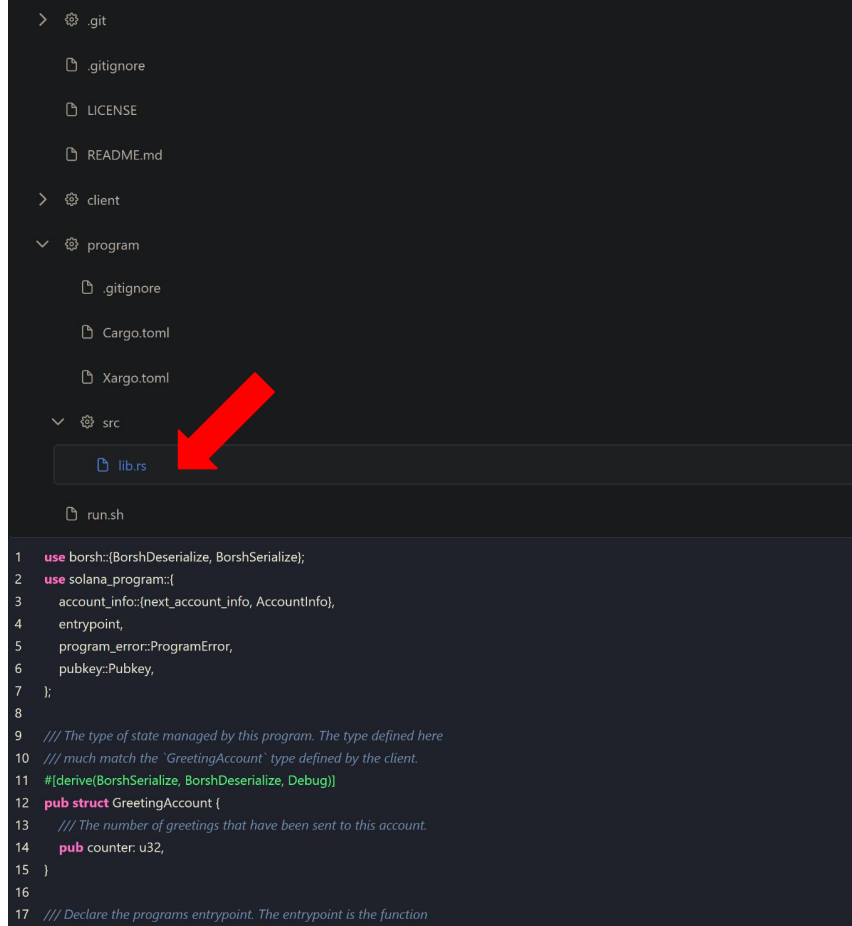
6. Tool outputs are stored in separate JSON files
7. Javascript file reads through JSON files and prints output to website



Demo Pictures



Initial page: Enter GitHub URL repo to be analyzed and choose tools.



```
1 use borsh::{BorshDeserialize, BorshSerialize};
2 use solana_program::{
3     account_info::next_account_info, AccountInfo,
4     entrypoint,
5     program_error::ProgramError,
6     pubkey::Pubkey,
7 };
8
9 /// The type of state managed by this program. The type defined here
10 /// much match the 'GreetingAccount' type defined by the client.
11 #[derive(BorshSerialize, BorshDeserialize, Debug)]
12 pub struct GreetingAccount {
13     /// The number of greetings that have been sent to this account.
14     pub counter: u32,
15 }
16
17 /// Declare the programs entrypoint. The entrypoint is the function
```

This is the result:

> chrono

▼ nix

Out-of-bounds write in nix::unistd::getgrouplist

On certain platforms, if a user has more than 16 groups, the `nix::unistd::getgrouplist` function will call the libc `getgrouplist` function with a length parameter greater than the size of the buffer it provides, resulting in an out-of-bounds write and memory corruption. The libc `getgrouplist` function takes an in/out parameter `ngroups` specifying the size of the group buffer. When the buffer is too small to hold all of the requested user's group memberships, some libc implementations, including glibc and Solaris libc, will modify `ngroups` to indicate the actual number of groups for the user, in addition to returning an error. The version of `nix::unistd::getgrouplist` in nix 0.16.0 and up will resize the buffer to twice its size, but will not read or modify the `ngroups` variable. Thus, if the user has more than twice as many groups as the initial buffer size of 8, the next call to `getgrouplist` will then write past the end of the buffer. The issue would require editing `/etc/groups` to exploit, which is usually nonly editable by the root user. Upgrade to ^0.20.2, ^0.21.2, ^0.22.2, >=0.23.0

> regex

> time

> tokio

Scroll down to see file directory and view files, and scroll more for code analysis output.

Task Schedule - UI

TASK TITLE	TASK OWNER	DURATION (DAYS)	PCT OF TASK COMPLETE	WEEK 1 (2/14)					WEEK 2					WEEK 3					WEEK 4				
				M	T	W	R	F	M	T	W	R	F	M	T	W	R	F	M	T	W	R	F
User Interface																							
Basic code upload to web app (GitHub)	Olivia	4	100%																				
View code analysis results on web app	Olivia	5	100%																				
File directory tree for code uploaded	Kipp	3	100%																				
Display user's code on website	Kipp	1	100%																				
Let user choose which analysis tools to use	Arsh	2	50%																				

TASK TITLE	TASK OWNER	DURATION (DAYS)	PCT OF TASK COMPLETE	WEEK 5 (3/21)					WEEK 6					WEEK 7					WEEK 8					WEEK 9					WEEK 10				
				T	W	R	F	M	T	W	R	F	M	T	W	R	F	M	T	W	R	F	M	T	W	R	F	M	T	W	R	F	
Show code analysis results on top of their code		5	0%																														
View analysis report		3	0%																														
User profile page		3	0%																														
See past code submissions and history for repos		4	0%																														
More polished frontend		3	0%																														

UI correctly shows files and displays result, but still needs a lot of work for our website to prove more useful than downloading the tools yourself.

Task Schedule - Backend

[illegible][illegible][illegible]

Task Schedule - Phase 3

TASK TITLE	TASK OWNER	DURATION (DAYS)	PCT OF TASK COMPLETE	WEEK 1 (2/14)					WEEK 2					WEEK 3					WEEK 4				
				M	T	W	R	F	M	T	W	R	F	M	T	W	R	F	M	T	W	R	F
Code analysis																							
Run cargo-audit on code	Arsh	3	100%																				
Run Soteria (free version) on code	Kipp	4	50%																				
Run Clippy on code	Arsh	3	100%																				
Write script to combine tools	Arsh	5	50%																				
Send uploaded code to server	Joseph	2	100%																				
Send test report back to the web server	Olivia	2	50%																				

TASK TITLE	TASK OWNER	DURATION (DAYS)	PCT OF TASK COMPLETE	WEEK 5 (3/21)					WEEK 6					WEEK 7					WEEK 8				
				T	W	R	F		M	T	W	R	F	M	T	W	R	F	M	T	W	R	F
Highlight tools finding the same warnings as each other		5	0%																				

Most of the code analysis work is already done aside from adding more tools and combining results to improve user experience.

Design Validation

- Results in comparison to competing solutions.
 - Faster response from submission
 - Focused on Rust analysis
 - Testing will involve comparing the results and speed of response between our own solution and similar solutions online.
- Front-end user friendliness
 - Minimal screen clutter
 - Clear user controls
 - Testing will involve asking users unfamiliar with the product to attempt to analyze a given Git repo without direction and hearing how they approached the interface.
- Review and report response time
 - With large code bases
 - With multiple users uploading at once

Design Validation Cont.

- Historical data recall
 - Fast queries
 - Remembers enough of the past to be useful
 - Testing will be conducted by creating a new user account and submitting at least 3 different versions of the same repo. The historical data will then be queried to test for correctness and retrieval speed.
- Database security
 - Are usernames and passwords stored properly?
 - Is code history confidential?
 - Difficult to test, development will adhere to industry standards as best as possible.

Design Demonstration Plan

- Review and report response time
 - With large code bases
 - With multiple users uploading at once
- Live demonstration to the class
 - Allow members of the class to create accounts on the remotely hosted web service
 - Allow members of the class to submit git repositories concurrently and see code review results as they are processed
 - Sample repositories will be provided for students without their own rust code
- Demonstration of report-on-push
 - Push to our own rust repository which has previously been enrolled
 - Wait for email to arrive in target inbox with code report.

Project Management

- Tasks for each member are written down in a Google Doc
- Team members facilitate the distribution of tasks depending on everyone's strong suits
- Continual progress checks at weekly meetings
- We will continue to use Gantt chart and weekly reports, to ensure that the development is on schedule
- Continue to meet on Mondays, Wednesdays, and Sundays
- All finance decisions will be made by the whole team

Teamwork

- Arsh:** Project lead
Select code-review tools
Create backend scripts
- Kipp:** Work on backend systems and connectivity
Handle the communications between the local server and the Amazon Database
Assist in review script integration
- Joseph:** Develop backend web server and assist frontend development
Communicate with professor and TA about project details and questions
Deploy web app to AWS
- Olivia:** Developing the front end design
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.

Project Concerns

- Many of the tools take a long time to run which can result in large loading times
- Some of the tools display unreadable information on the frontend

Any Questions?

