QuillAudits

Audit Report
June, 2024

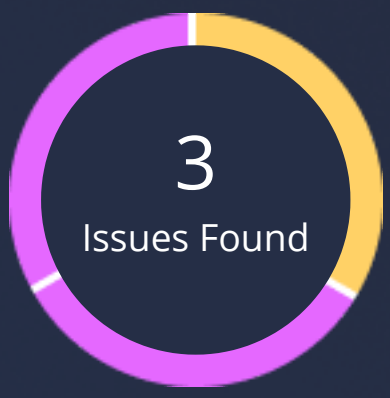
For

Table of Content

Executive Summary

Project Name	Ecobal
Overview	The audited program is designed to enhance security and maintainability in Solana token operations by introducing a proxy program between the client and the token program. Presented architecture provides additional control for verifying permissions and simplifies future updates to the token infrastructure.
Timeline	1st June 2024 - 3rd June 2024
Updated Code Received	12th June 2024
Second Review	12th June 2024
Method	Manual Review, Functional Testing, Automated Testing, etc. All the raised flags were manually reviewed and re-tested to identify any false positives.
Audit Scope	The scope of this audit was to analyse the Ecobal Contract for quality, security, and correctness.
Source Code	.rs File Provided by Ecobal team
Contracts In-Scope	ecobal_token
Branch	NA
Contracts out of Scope	In-scope contracts have been audited by QuillAudits. However, these contracts inherit functionality from out-of-scope Smart contracts that were not audited. Vulnerabilities in unaudited contracts could impact in-scope Smart Contracts functionality. QuillAudits is not responsible for such vulnerabilities. Below are Out of Scope Contracts: Solana native token program, Anchor framework
Fixed In	main (ee1259a)

Number of Security Issues per Severity



High
Medium
Low
Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	0	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	1	0	2

Checked Vulnerabilities

We have scanned the solana program for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Signer authorization
- Account data matching
- Sysvar address checking
- Owner checks
- Type cosplay
- Initialization
- Arbitrary cpi
- Duplicate mutable accounts
- Bump seed canonicalization
- PDA Sharing
- Incorrect closing accounts
- Missing rent exemption checks
- Arithmetic overflows/underflows
- Numerical precision errors
- Solana account confusions
- Casting truncation
- Insufficient SPL token account verification
- Signed invocation of unverified programs

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of Token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Hardhat, Foundry.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impact and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Medium Severity Issues

1. Potential Phishing Vulnerability in Proxy Architecture Due to Lack of Underlying Token Verification

Path

programs/ecobal_token/src/lib.rs

Description

It was observed that the current architecture of the proxy program can be exploited for phishing attempts. The proxy program does not verify the underlying token involved in its operations. This lack of verification allows malicious actors to create instructions for any token. Consequently, a valid token transfer instruction could be manipulated to interact with other tokens, such as transferring USDC instead of the intended Ecobal token. Such phishing attempts can be successful because user can assume that they are interacting with Ecobal tokens - as the program address indicates.

Recommendation

1. Initialize Program with Token Address: Modify the proxy program to require the initialization with a specific token address.
2. Token Address Validation: Implement a validation step in each instruction to ensure the token address matches the initialized address before executing any operations (transfer, mint, burn, etc.).
3. Error Handling: Reject any instructions involving a token address that does not match the initialized address, and return an appropriate error message.

Status

Resolved

Resolved in Commit Hash

ee1259a5aa4a2ed4536cb52dd75230cd23a8766a

Informational Issues

1. Missing Documentation and README

Description

The Solana program lacks comprehensive documentation and a README file, which are essential for developers and users to understand and effectively utilize the program. Without proper documentation, it becomes challenging to grasp the program's functionalities, deployment process, and usage guidelines. This can lead to misconfigurations, improper usage, and a steeper learning curve for new contributors and users.

Recommendation

To address this issue, create detailed documentation and a README file.

Status

Resolved

Resolved in Commit Hash

ee1259a5aa4a2ed4536cb52dd75230cd23a8766a

2. Improper Anchor Environment Configuration

Path

Ecobal program

Description

The current setup of the Solana program using the Anchor framework is incomplete due to a mismatch of programId in Anchor.toml and lib.rs, which causes all tests to fail. Additionally, the Anchor.toml configuration points to the developer's private key as the provider wallet, which is not appropriate for deployment or testing purposes. Proper configuration of the Anchor environment is crucial for deploying and interacting with the program on the blockchain.

Recommendation

1. ProgramId Consistency: Ensure that the programId specified in Anchor.toml matches the programId defined in lib.rs.
2. Provider Wallet Configuration: Properly configure the provider wallet in Anchor.toml by defining a valid wallet address. For local tests, generate new keys specifically for testing and upload them to the project repository. These keys should not be used in normal operations.

Status

Resolved

Resolved in Commit Hash

ee1259a5aa4a2ed4536cb52dd75230cd23a8766a

Functional Tests Cases

1. Token Transfer

Function: proxy_transfer

Test Case: Verify Token Transfer Functionality

- Objective: Ensure that the proxy_transfer function can handle token transfers correctly.
- Steps:
  - Initialize the ProxyTransfer context with valid sender and receiver accounts.
  - Call proxy_transfer with a specified amount.
  - Check the sender's balance to confirm it decreased by the transferred amount.
  - Check the receiver's balance to confirm it increased by the transferred amount.
- Expected Results:
  - Transfer is successfully completed.
  - Correct balances are updated.
  - TokenTransferEvent is emitted.

2. Mint Tokens

Function: proxy_mint_to

Test Case: Verify Token Minting Functionality

- Objective: Ensure that the proxy_mint_to function can mint new tokens correctly.
- Steps:
  - Initialize the ProxyMintTo context with a valid account.
  - Call proxy_mint_to with a specified amount.
  - Check the account balance to confirm it increased by the minted amount.
- Expected Results:
  - Tokens are minted successfully.
  - Account balance is updated correctly.
  - TokenMintEvent is emitted.

3. Burn Tokens

Function: proxy_burn

Test Case: Verify Token Burning Functionality

- Objective: Ensure that the proxy_burn function can burn tokens correctly.
- Steps:
  - Initialize the ProxyBurn context with a valid account.
  - Call proxy_burn with a specified amount.
  - Check the account balance to confirm it decreased by the burned amount.
- Expected Results:
  - Tokens are burned successfully.
  - Account balance is updated correctly.
  - TokenBurnEvent is emitted.

4. Freeze Account

Function: proxy_freeze_account

Test Case: Verify Account Freezing Functionality

- Objective: Ensure that the proxy_freeze_account function can freeze an account correctly.
- Steps:
  - Initialize the ProxyFreezeAccount context with a valid account.
  - Call proxy_freeze_account.
  - Attempt to perform a token transfer or mint from the frozen account.
- Expected Results:
  - Account is frozen successfully.
  - Transfers or mints from the frozen account are blocked.
  - AccountFreezeEvent is emitted.

5. Thaw Account

Function: proxy_thaw_account

Test Case: Verify Account Thawing Functionality

- Objective: Ensure that the proxy_thaw_account function can thaw an account correctly.
- Steps:
  - Initialize the ProxyThawAccount context with a previously frozen account.
  - Call proxy_thaw_account.
  - Attempt to perform a token transfer or mint from the thawed account.
- Expected Results:
  - Account is thawed successfully.
  - Transfers or mints from the thawed account are allowed.
  - AccountThawEvent is emitted.

6. Close Account

Function: proxy_close_account

Test Case: Verify Account Closure Functionality

- Objective: Ensure that the proxy_close_account function can close an account correctly.
- Steps:
  - Initialize the ProxyCloseAccount context with a valid account.
  - Call proxy_close_account.
  - Check that the account is closed and any remaining balance is transferred to a specified account.
- Expected Results:
  - Account is closed successfully.
  - Remaining balance is transferred correctly.
  - AccountCloseEvent is emitted.

7. Set Authority

Function: proxy_set_authority

Test Case: Verify Authority Setting Functionality

- Objective: Ensure that the proxy_set_authority function can set new authorities correctly.
- Steps:
  - Initialize the ProxySetAuthority context with valid accounts.
  - Call proxy_set_authority with a specified authority type and new authority.
  - Verify that the new authority is set correctly.
- Expected Results:
  - Authority is set successfully.
  - Operations requiring the new authority are authorized correctly.
  - AuthoritySetEvent is emitted.

8. Approve Delegation

Function: proxy_approve

Test Case: Verify Delegation Approval Functionality

- Objective: Ensure that the proxy_approve function can approve token delegation correctly.
- Steps:
  - Initialize the ProxyApprove context with valid accounts.
  - Call proxy_approve with a specified amount.
  - Verify that the delegation is approved correctly.
- Expected Results:
  - Delegation is approved successfully.
  - Delegate can perform approved operations up to the specified amount.
  - DelegationApprovalEvent is emitted.

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of the Ecobal codebase. We performed our audit according to the procedure described above.

Some issues of Medium, Low and informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in Ecobal smart contracts. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of Ecobal smart contracts. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of the Ecobal to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

1000+
Audits Completed

$30B
Secured

1M+
Lines of Code Audited

Follow Our Journey

Audit Report
June, 2024

For

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com