QuillAudits

Audit Report
July, 2024

For

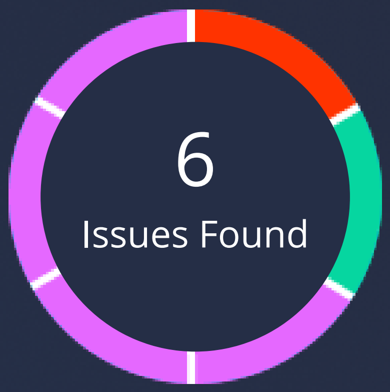
OGLONG

Table of Content

Executive Summary

Project Name	OgLong
Overview	Oglong Bridge is a decentralized application (dApp) built on the Solana blockchain, designed to manage token minting, and burning operations securely. It ensures that only authorized accounts can perform these operations, maintaining the integrity and security of user-owned token accounts.
Timeline	19th July 2024 to 20th July 2024
Updated Code Received	22nd July 2024
Second Review	22nd July 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 OgLong Programs for quality, security, and correctness.
Source Code	https://gitlab.com/brc-group/solana-program/-/tree/staging/programs/oglong-bridge/src?ref_type=heads
Contracts In-Scope	oglong-bridge
Branch	Staging (6d7c6d0d5fa370c3b27bfc445e55d6ec284b86dc)
Fixed In	0aa8451d7ff4305530d4b8021a29231d39b5d280

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	1
Partially Resolved Issues	0	0	0	0
Resolved Issues	1	0	1	3

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 ERC's 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.

High Severity Issues

1. Improper Token Validation in Burn Function

Path

programs/oglong-bridge/src/lib.rs

Function

og_burn

Description

The current implementation of the Burn function does not validate if the user is attempting to burn the correct token. This flaw allows users to burn any token, including valueless ones, which is then accounted for by the off-chain logic. Consequently, this can lead to significant financial loss as users could receive tokens on the other side of the bridge without burning the correct tokens.

Recommendation

Implement proper token validation within the Burn function to ensure that only the specific bridge token can be burned. This can be achieved by checking the token's mint address and ensuring it matches the bridge token's mint address before proceeding with the burn operation. Additionally, add comprehensive unit tests to verify the correct implementation and prevent similar issues in the future.

Status

Resolved

Low Severity Issues

2. Insufficient Validation in SetOwner Function

Path

programs/oglong-bridge/src/lib.rs

Function

set_owner

Description

The SetOwner function currently lacks proper validation to ensure that the new owner is the same as the og_mint authority. Without this validation, it is possible to set an inappropriate owner, which could break the program and disrupt its intended operations.

Recommendation

Modify the SetOwner function to include a validation check that ensures the new owner is the same as the og_mint authority. This can be done by comparing the provided new owner address with the og_mint authority address before completing the owner change. Additionally, implement unit tests to verify this validation logic and prevent future issues.

Status

Resolved

Informational Issues

3. Use of msg! Instead of Solana Anchor Events

Path

programs/oglong-bridge/src/lib.rs

Function

*

Description

The current implementation uses msg! for logging and passing transaction strings to the mint and burn functions. This approach is not CU (Gas) optimized and is not optimal for tracking and deserializing events, which can complicate off-chain indexing and monitoring.

Recommendation

Switch to using Solana Anchor events instead of msg! for logging and passing transaction information. Anchor events are more robust and can be easily tracked and deserialized, which will simplify off-chain indexing and improve the overall maintainability of the code. Implement Anchor event handlers in the mint and burn functions and update the off-chain indexer to process these events.

Status

Resolved

4. Missing Event Emission for Critical Actions

Path

programs/oglong-bridge/src/lib.rs

Function

*

Description

Critical actions such as initialization and setting the owner do not emit events. The absence of events for these actions can lead to difficulties in tracking and auditing these operations off-chain, reducing transparency and traceability.

Recommendation

Implement event emission for critical actions like initialization and setting the owner. Use Solana Anchor events to log these actions, ensuring that each event includes relevant details such as the action performed, the involved accounts, and timestamps. This will improve off-chain tracking and auditing capabilities.

Status

Resolved

5. Use of Developer’s Own Wallet in Anchor.toml

Path

Anchor.toml

Function

*

Description

The Anchor.toml configuration file points to the developer’s own wallet, which prevents others from running tests without modifying this file. This practice hinders collaboration and makes automated testing setups more difficult.

Recommendation

Configure Anchor.toml to use a generic or environment-based wallet setup, avoiding hardcoding a specific developer’s wallet. This can be achieved by setting the provider.wallet to an environment variable, allowing each developer to use their own wallet without modifying the configuration file. Additionally, update documentation to guide developers on how to set up their environment variables for wallet configuration.

Status

Resolved

6. Missing Test Coverage

Path

Anchor.toml

Function

*

Description

The project lacks adequate test coverage, making it difficult to ensure the reliability and security of the program. The absence of tests increases the risk of undetected bugs and vulnerabilities.

Recommendation

Develop a comprehensive suite of tests covering all critical functions and edge cases of the program. Include unit tests for individual functions like minting, burning, and setting the owner, as well as integration tests to validate the overall program behavior. Utilize Solana's testing framework and ensure tests can be run in any development environment without additional configuration.

Status

Acknowledged

Functional Tests Cases

- ✓ Initialize Configuration for Token Minting
- ✓ Mint Tokens to User Accounts
- ✓ Burn Tokens from User Accounts
- ✓ Set New Owner for Program

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 OgLong 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. In the end, OgLong Team Resolved all Issues.

Disclaimer

About QuillAudits

QuillAudits is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers. With six years of expertise, we've secured over 1000 projects globally, averting over $30 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.

1000+
Audits Completed

$30B
Secured

1M+
Lines of Code Audited

Follow Our Journey

Audit Report
July, 2024

For

OGLONG

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com