SMART CONTRACT AUDIT

June 8th 2023 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges.

TECHNICAL SUMMARY

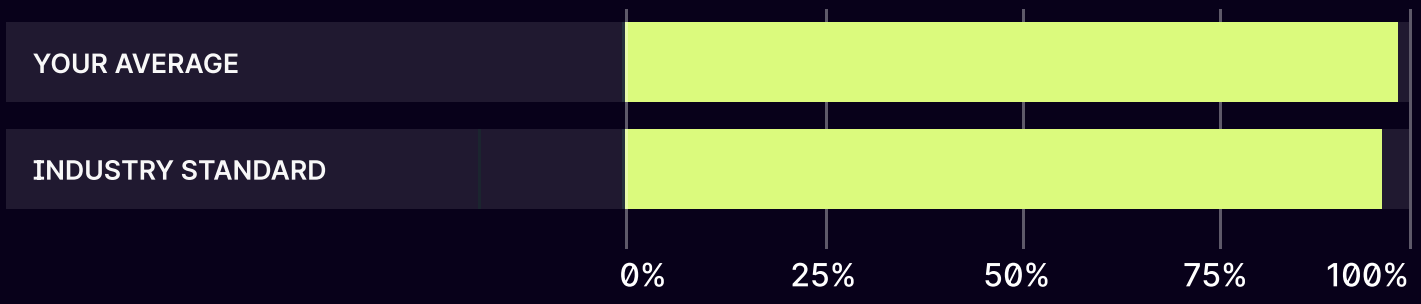
This document outlines the overall security of the Aurora smart contracts evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the Aurora smart contract codebase for quality, security, and correctness.

Contract Status

There was no critical issue found during the audit. (See Complete Analysis)

Testable Code



100% of the code is testable, which is above the industry standard of 95%.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the NEAR network's fast-paced and rapidly changing environment, we recommend that the Aurora team put in place a bug bounty program to encourage further active analysis of the smart contract.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the Aurora repository: https://github.com/aurora-is-near/rainbow-token-connector

Last commit - https://github.com/aurora-is-near/rainbow-token-connector/pull/166/files

Within the scope of this audit, the team of auditors reviewed the following contract(s):

- Bridge-common
- Token-locker

During the audit, Zokyo Security ensured that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most resent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo's Security Team has followed best practices and industry-standard techniques to verify the implementation of Aurora smart contracts. To do so, the code is reviewed line-by-line by our smart contract developers, documenting any issues as they are discovered. Part of this work includes writing a unit test suite using the near-workspace testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01	Due diligence in assessing the overall code quality of the codebase.	03	Testing contract logic against common and uncommon attack vectors.
02	Cross-comparison with other, similar smart contracts by industry leaders.	04	Thorough manual review of the codebase line by line.

Executive Summary

Throughout the audit process, there were no notable problems identified. However, we did come across one issue of low severity and a few informational concerns. It is crucial to highlight that these findings might only affect certain circumstances involving the contract owner and the investors involved. For more in-depth insights, kindly refer to the "Complete Analysis" section.

STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

FINDINGS SUMMARY

#	Title	Risk	Status
1	Calling `contains` before `insert`	Low	Acknowledged
2	Code duplication	Informational	Acknowledged
3	Unimplemented functionality	Informational	Resolved

Calling `contains` before `insert`

File: token-locker/src/lib.rs
Function: Contract::record_proof
Details:
The `UnorderedSet::insert` function returns **false** if the element already exists in the set. While the `contains` call requires less gas than `insert` itself it is still better not to use it in this case. This way it makes a little cheaper transactions with duplicated proofs, while the normal transactions become a little more expensive (for the excess `contains` call)

Recommendation:
try the construction below instead:

```
assert!(
self.used_events.insert(&proof_key),
"Event cannot be reused for withdrawing."
);
```

INFORMATIONAL | ACKNOWLEDGED

Code duplication

File: token-locker/src/lib.rs
Trait: ExtToken
Details:
The given trait declares the functions that are defined by the NEP141 standard + MetadataProvider. It could be better to import those traits from the **"near-contract-standards"** library instead.

Recommendation:
use the construction like *"pub trait ExtToken: FungibleTokenCore + FungibleTokenMetadataProvider {}"*

INFORMATIONAL | RESOLVED

Unimplemented functionality

File: token-locker/src/whitelist.rs
Function: Contract::check_whitelist_token
Details:
The function checks the **`is_whitelist_mode_enabled`** value of the Contract, which is always **true.** There is no functionality to switch it to be **false.**

Recommendation:
Either remove the check with the flag or add a mode switching function

	Bridge-common	Token-locker
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Security

As a part of our work assisting Aurora in verifying the correctness of their contract code, our team was responsible for writing integration tests using the near-workspace testing framework.

The tests were based on the functionality of the code, as well as a review of the Aurora contract requirements for details about issuance amounts and how the system handles these.

Unit test for: bridge-common
Finished test [unoptimized + debuginfo] target(s) in 0.17s
Running unittests src/lib.rs (target/debug/deps/bridge_common-66864974c31ad55f)

running 9 tests
✓ test prover_test::test_ ... ok
✓ test prover_test::test_validate_eth_address ... ok
✓ test lib_test::test_parse_recipient ... ok
✓ test result_types_test::initialized_test ... ok
✓ test prover_test::test_proof_get_key ... ok
✓ test result_types::generate_result_prefixs ... ok
✓ test result_types_test::generate_result_prefixs ... ok
✓ test prover_test::test_validate_eth_address_panic_invalid_bytes - should panic ... ok
✓ test prover_test::test_validate_eth_address_panic_invalid_hex - should panic ... ok

test result: ok. 9 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s

Unit test for: token-locker
Finished test [unoptimized + debuginfo] target(s) in 45.69s
Running unittests src/lib.rs (target/debug/deps/rainbow_bridge_near_token_locker-b8e92c556e09f966)

running 31 tests
✓ test lib_test::test_update_factory_address ... ok
✓ test lib_test::test_log_metadata ... ok
✓ test tests::test_account_in_whitelist ... ok
✓ test lib_test::test_is_used_proof ... ok
✓ test lib_test::test_lock_unlock_token ... ok
✓ test lib_test::test_only_admin_can_pause ... ok

✓ test tests::test_lock_unlock_token ... ok
✓ test tests::test_only_admin_can_pause ... ok
✓ test tests::test_account_not_in_whitelist - should panic ... ok
✓ test lib_test::test_finish_withdraw_exist_event - should panic ... ok
✓ test token_receiver_test::test_lock_unlock_token ... ok
✓ test unlock_event_test::test_fmt ... ok
✓ test tests::test_tokens_in_whitelist ... ok
✓ test whitelist_test::test_add_account_to_whitelist ... ok
✓ test tests::test_remove_account_from_whitelist - should panic ... ok
✓ test tests::test_accounts_in_whitelist ... ok
✓ test lib_test::test_withdraw - should panic ... ok
✓ test tests::test_token_not_in_whitelist - should panic ... ok
✓ test whitelist_test::test_check_whitelist_token_mode ... ok
✓ test whitelist_test::test_add_account_to_whitelist_not_set - should panic ... ok
✓ test whitelist_test::test_check_whitelist_token ... ok
✓ test tests::test_blocked_token - should panic ... ok
✓ test whitelist_test::test_get_token_account_key ... ok
✓ test whitelist_test::test_check_whitelist_token_panic_block - should panic ... ok
✓ test whitelist_test::test_remove_account_from_whitelist ... ok
✓ test whitelist_test::test_check_whitelist_token_panic_account - should panic ... ok
✓ test whitelist_test::test_set_token_whitelist_mode ... ok
✓ test lib_test::test_finish_withdraw_unused_some ... ok
✓ test whitelist_test::test_add_account_to_whitelist_panic - should panic ... ok
✓ test unlock_event_test::fuzzing_eth_unlocked ... ok
✓ test unlock_event::tests::fuzzing_eth_unlocked ... ok

test result: ok. 31 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.22s

The resulting code coverage (i.e., the ratio of tests-to-code) is as follows:

FILE	% STMTS	% BRANCH	% FUNCS	% FUNCS	UNCOVERED LINES
Token-locker	100	100	23.63	98.35	
Bridge-common	100	100	16.98	45.07	
All Files	**100**	**100**	**20.30**	**71.71**	

We are grateful for the opportunity to work with the Aurora team.

The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.

Zokyo Security recommends the Aurora team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo