Blaize.Security

June 7th, 2022 / V. 1.0

AURORA

SMART CONTRACT AUDIT

TABLE OF CONTENTS

AUDIT RATING

Aurora contract's source code was taken from the repository provided by the Aurora team.

SCORE **9.9** /10

The scope of the project is **Aurora** set of contracts**:**

1/ Treasury.sol

2/ AdminControlled.sol

3/ JetStakingV1.sol

Repository:
https://github.com/aurora-is-near/aurora-staking-contracts

Initial commit

- 45f79c9ddcf8112d3dd4f1f31a9c4354dbb529e1

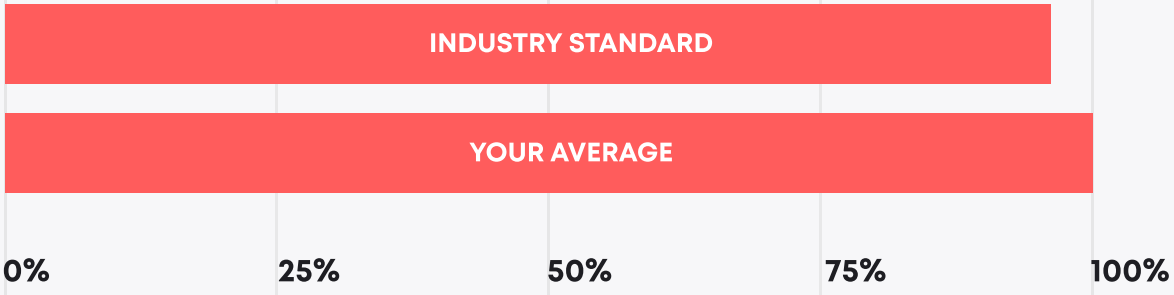
Last audited commit

- e32dc4197bd3cb4db4178695e969f58b053821b3
  develop branch

TECHNICAL SUMMARY

In this report, we consider the security of the contracts for Aurora protocol. Our task is to find and describe security issues in the smart contracts of the platform. This report presents the findings of the security audit of **Aurora** smart contracts conducted between **May 2nd, 2022 - June 7th, 2022.**

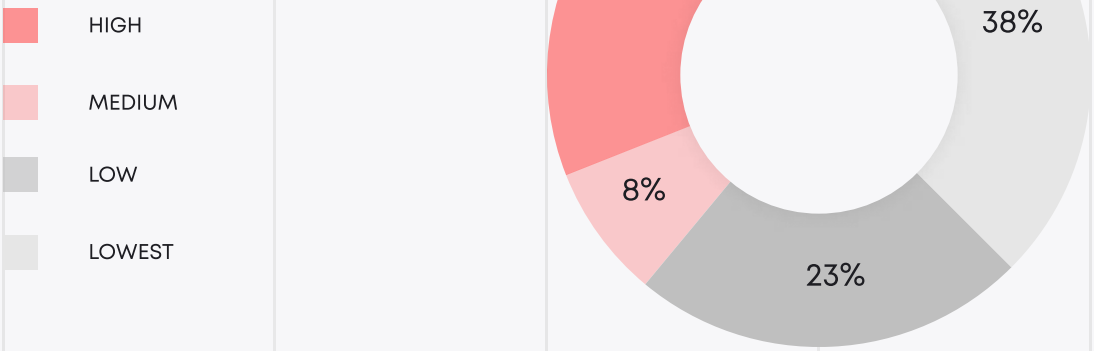
Testable code



The testable code is 99,45%, which corresponds to the industry standard of 95%.

The scope of the audit includes the unit test coverage, that bases on the smart contracts code, documentation and requirements presented by the Aurora team. Coverage is calculated based on the set of Truffle framework tests and scripts from additional testing strategies. Though, in order to ensure a security of the contract Blaize.Security team recommends the Aurora team put in place a bug bounty program to encourage further and active analysis of the smart contracts.

THE GRAPH OF VULNERABILITIES DISTRIBUTION:



The table below shows the number of found issues and their severity. A total of 13 problems were found. 12 issues were fixed or verified by the Aurora team.

	FOUND	FIXED/VERIFIED
Critical	0	0
High	4	4
Medium	1	0
Low	3	3
Lowest	5	5

SEVERITY DEFINITION

Critical

A system contains several issues ranked as very serious and dangerous for users and the secure work of the system. Needs immediate improvements and further checking.

High

A system contains a couple of serious issues, which lead to unreliable work of the system and migh cause a huge information or financial leak. Needs immediate improvements and further checking.

Medium

A system contains issues which may lead to mediumfinancial loss or users' private information leak. Needs immediate improvements and further checking.

Low

A system contains several risks ranked as relatively small with the low impact on the users' information and financial security. Needs improvements.

Lowest

A system does not contain any issue critical to the secure work of the system, yet is relevant for best

AUDITING STRATEGYAND TECHNIQUES APPLIED \ PROCEDURE

We have scanned this smart contract for commonly known and more specific vulnerabilities:

- Unsafe type inference;
- Timestamp Dependence;
- Reentrancy;
- Implicit visibility level;
- Gas Limit and Loops;
- Transaction-Ordering Dependence;
- Unchecked external call - Unchecked math;
- DoS with Block Gas Limit;
- DoS with (unexpected) Throw;
- Byte array vulnerabilities;
- Malicious libraries;
- Style guide violation;
- ERC20 API violation;
- Uninitialized state/storage/ local variables;
- Compile version not fixed.

Procedure

In our report we checked the contract with the following parameters:

- Whether the contract is secure;
- Whether the contract corresponds to the documentation;
- Whether the contract meets best practices in efficient use of gas, code readability;

Automated analysis:

Scanning contract by several public available automated analysis tools such as Mythril, Solhint, Slither and Smartdec. Manual verification of all the issues found with tools.

Manual audit:

Manual analysis of smart contracts for security vulnerabilities. Checking smart contract logic and comparing it with the one described in the documentation.

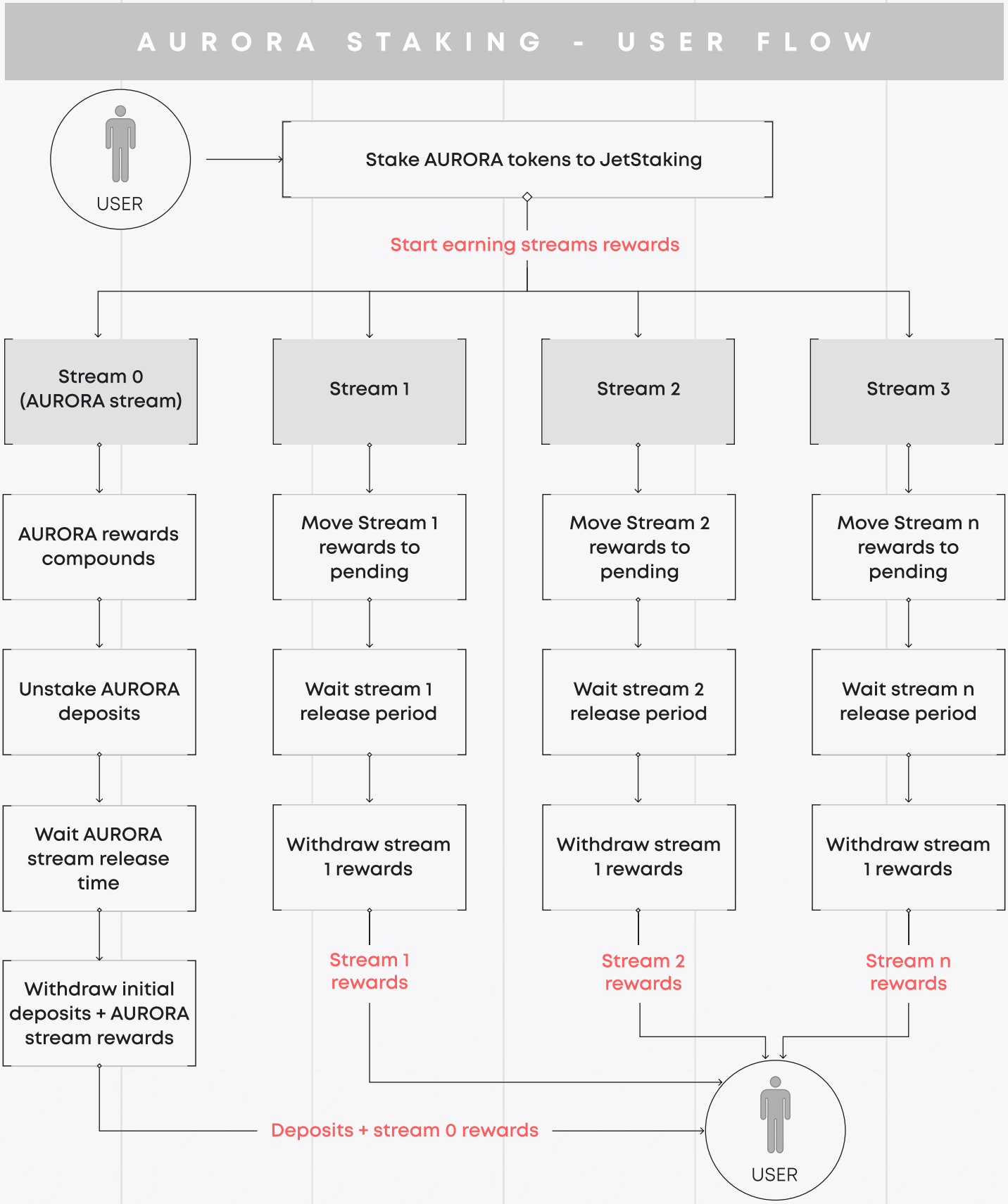
EXECUTIVE SUMMARY

The contract contained several high risk issues connected to the incorrect funds flow, incorrect streams handling and unclear staking program. Though, the team has fixed these issues.
All other issues were connected to missed checks and validations, clarifications about the admin role functionality, and correct flow of ETH receiving. Nevertheless, all security risk issues were fixed by the team.

The overall security is high, code is well documented, has good native tests coverage. Auditor's team has carefully reviewed smart contract's business logic, provided several rounds of testing and verified the correctness of native protocol tests.

	RATING
Security	9.7
Gas usage and logic optimization	9.9
Code quality	10
Test coverage	10
Total	9.9

PROTOCOL OVERVIEW



COMPLETE ANALYSIS

HIGH ✔ Resolved

Deprecated Eth transfer

AdminControlled.sol: function adminSendEth().
Due to the Istanbul update there were several changes provided to the EVM, which made .transfer() and .send() methods deprecated for the ETH transfer. Thus it is highly recommended to use .call() functionality with mandatory result check, or the built-in functionality of the Address contract from OpenZeppelin library. This should be done in order to mitigate any possible future update of EVM for the Aurora network

Recommendation:

Correct ETH sending functionality.

Post-audit:

Functionality of receiving or transferring ETH was removed.

HIGH ✔ Resolved

Missing validation that stream is not already activated.

JetStakingV1: function cancelStreamProposal().
Currently there are validations that schedule has started and stream is proposed in the function. However, the function is missing validation, that the stream wasn't actually activated by the stream creator.

Recommendation:

Add a validation that the stream wasn't activated.

Post-audit:

A flag was added to stream struct, which signalisies about the state of stream. Only streams in state PROPOSED can be canceled.

HIGH	✔ Resolved

Incorrect reward schedule update.

JetStakingV1.sol

1. Function proposeStream() should validate that parameter “maxDepositAmount” == summation of values from array “scheduleRewards” to make sure that stream owner will send enough amount of reward token.
2. In case, stream owner sends less reward amount than “maxDepositAmount”, the reward schedule is updated in function _updateStreamRewardSchedules(). Currently, the function doesn’t perform accurate calculation.

Example:

1. reward schedule = [30, 20, 10, 0], maxDepositAmount = 60
2. Stream owner sends 40 tokens instead of 60. rewardTokenAmount = 40
3. Function recalculates new reward schedule, where suggestedAmount = rewardSchedule[0] (Line 1128)

- rewardSchedule[0] = rewardSchedule[0] * rewardTokenAmount / suggestedAmount = 30 * 40 / 30 = 40.
- rewardSchedule[1] = rewardSchedule[1] * rewardTokenAmount / suggestedAmount = 20 * 40 / 30 = 20 * 40 / 30 = 26.6.
- rewardSchedule[2] = rewardSchedule[2] * rewardTokenAmount / suggestedAmount = 10 * 40 / 30 = 13.3.

Total schedule reward amount = 40 + 26.6 + 13.3 ~ 80, while stream owner sent only 40 tokens.

Recommendation:

In order to recalculate the new reward schedule correctly, suggestedAmount should be equal to previous summation of rewards or “maxDepositAmount”. In this case, results of steps a-c in example would be equal:

suggestedAmount = maxDepositAmount = 60.

- rewardSchedule[0] = rewardSchedule[0] * rewardTokenAmount / suggestedAmount = 30 * 40 / 60 = 20.

- rewardSchedule[1] = rewardSchedule[1] * rewardTokenAmount / suggestedAmount = 20 * 40 / 60 = 20 * 40 / 30 = 13.3.
- rewardSchedule[2] = rewardSchedule[2] * rewardTokenAmount / suggestedAmount = 10 * 40 / 60 = 6.6.

Total schedule reward amount = 20 + 13.3 + 6.6 ~ 40 which is a new rewardTokenAmount.

Post-audit:

A require statement was added in _validateStreamParameters(), which validates that “maxDepositAmount” equals to rewardSchedule[0] instead of a sum of all rewardSchedules, which forbids owner to deposit the amount, necessary for all reward schedules. Based on the logic of the contract, maxDepositAmount should be equal to summation of all values from the rewardSchedule array.

It was verified that the first value from the array is a total amount of reward for stream, which makes calculations accurate. Verifying that “maxDepositAmount” equals to rewardSchedule[0] is a correct check.

HIGH	✔ Resolved

Variable admin is not initialized.

AdminController.sol

Variable “admin” is not initialized during the call of initializer function __AdminControlled_init(). The only way to initialize the variable is an additionally call of transferOwnership(). The issue is marked as High since in case transferOwnership() wasn’t called on time, any tokens might potentially be transferred to the zero address in contract JetStakingV1.sol(Lines 278, 311, 356, 364).

Recommendation:

Initialize variable “admin” during execution of function __AdminControlled_init()

Post-audit:

Variable “admin” was removed from the contract. Any tokens, necessary to be sent back, are now sent to the address“manager” from struct Stream, which is assigned to msg.seder when creating the stream.

MEDIUM

✔ **Aknowledged**

Variable admin is not initialized.

JetStakingV1.sol: function _before() (Line 934), function _moveAllRewardsToPending() (Line 991), function stake() (1042). Iteration through a storage array might consume more gas than allowed per transaction, thus functions will always revert. Loop will also iterate through canceled streams or streams where the reward schedule is finished.

Recommendation:

Consider removing streams from the array. Streams, which are canceled in functions cancelStreamProposal(), removeStream() can be removed from an array. An additional function can be added to remove streams whose reward schedules are finished.

Post-audit:

Client is aware of this issue. It is agreed that there won't be more than 10 streams currently. The issue will be fixed later, including the optimization of the formula of RPS calculations.

LOW

✔ **Resolved**

Validate that reward token is whitelisted.

JetStakingV1: function _validateStreamParameters().
Contract Treasury.sol supports only whitelisted tokens(For example, when paying rewards), so provided "rewardToken" should be validated to be supported during the proposing of the stream.
Issue is marked as low, since only the admin can create a proposal for a stream, however the validation should still be added to the contract.

Recommendation:

Validate that Treasury supports "rewardToken".

LOW ✔ Resolved

Missing visibility identifier.

JetStakingV1: missing visibility for ONE_MONTH, FOUR_YEARS, RPS_MULTIPLIER, maxWeight, minWeight, users, streams.

Recommendation:

Set variable visibility.

LOW ✔ Resolved

Add a minimum period of time between current timestamp and the start of schedule.

JetStakingV1: function _validateStreamParameters().
Currently, the start of rewards schedule must be greater than current block.timestamp(Validation on Line 1095), thus, it is possible to pass scheduleTimes[0] with low difference with current timestamp, so that the creator of stream will not have enough time to activate the stream.

Recommendation:

Add a minimum period of time between scheduleTimes[0] and block.timestamp to make sure the stream creator has enough time to activate the stream.

LOWEST	✔ Resolved

receive() can be defined instead of a function.

AdminControlled.sol: function adminReceiveEth().
A receive() function can be defined instead of a payable function with an empty body to allow a contract to receive Eth. Also, based on the name of the function(started with word “admin”) function might be restricted to be executed only by admin.

Recommendation:

Use receive() instead of a function with an empty body to receive Eth. Example: receive() external payable {}.
Verify, that function should not be restricted.

Post-audit:

Functionality of receiving ETH was removed.

LOWEST	✔ Resolved

Validate “tau” parameter.

JetStakingV1: function _validateStreamParameters().
Parameter “tau” should be validated in order not to be equal to extremely big values, due to which users won’t be able to withdraw their pending rewards.

Recommendation:

Validate that “tau” is not equal to big value and doesn’t block users from withdrawing their pending rewards.

LOWEST	✔ Verified

Admin is able to provide any fund receiver when removing a stream.

JetStakingV1: function removeStream().
The rest of the rewards are moved to the address "streamFundReceiver" which is not necessary to be a stream owner. However, the comment section before transferring rewards (Line 363) says that the rest of rewards are transferred to the creator of the stream.

Recommendation:

Either confirm that admin has to provide the address of the receiver of rewards or transfer the rest of the rewards to stream owner address which is stored in Stream struct.

From client:

Admin should be able to provide an arbitrary address for receiving funds. Documentation in the contract is updated to correspond this.

LOWEST	✔ Verified

Delegate call to an arbitrary address.

AdminControlled.sol: function adminDelegatecall().
Function performs delegate call to an arbitrary address, which can cause unknown effects. For example, storage variables might be corrupted and prevent protocol from operating correctly or tokens can be withdrawn from treasury. Issue is marked as info since only an admin can call this function, however admin rights are supposed to be transferred to DAO, which will be driven by a community. This can potentially be used by malicious actors.

Recommendation:

Either remove delegate call to an arbitrary address or add a whitelist of address to which delegate call can be performed.

Post-audit:

Client verified that function is required by the contract and cannot be removed. Whitelisting of target addresses will happen on the side of admin. Client is also aware of the risks which might take place in case admin rights are granted to DAO contract.

LOWEST	✔ Verified

Admin controls treasury rewards.

Treasury.sol
It needs to be reflected in the report, that the admin has full control of the treasury, thus he decides of the amounts of tokens withdrawn from the treasury.

Recommendation:

Add restrictions or verify that the ownership will transferred to the DAO.

From client:

Moving tokens to DAO tokens would be the next step in increasing the security of funds.

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

	AdminControlled.sol	JetStakingV1.sol
✓ Re-entrancy	Pass	Pass
✓ Access Management Hierarchy	Pass	Pass
✓ Arithmetic Over/Under Flows	Pass	Pass
✓ Delegatecall Unexpected Ether	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 BY AURORA TEAM

Contract: AdminControlled

- ✓ should admin able to pause the contract (422ms)
- ✓ should pause role able to pause the contract and only admin can unpause the contract (234ms)
- ✓ should allow admin to change the storage layout using admin SSTORE (103ms)
- ✓ should allow admin to change SSTORE with mask (94ms)
- ✓ should allow admin to delegate call (146ms)

Contract: JetStakingV1

- ✓ should test multiple stakers reward calculation (1867ms)
- ✓ should test multiple stakers compound reward during 6 months (520ms)
- ✓ should test multiple stakers compound reward during 1 year (875ms)
- ✓ should return treasury account
- ✓ should allow admin to propose new stream (243ms)
- ✓ should allow stream owner to create a stream (451ms)
- ✓ should refund stream owner when stream created with less rewards (528ms)
- ✓ should create stream and refund staking admin if deposit reward is less than the upper amount (680ms)
- ✓ should release aurora rewards to stream owner (870ms)
- ✓ should stake aurora tokens (153ms)

- ✓ should not release new rewards in the same block (196ms)
- ✓ user stakes and never claims (345ms)
- ✓ should able to get schedule times per stream
- ✓ should be able to get reward per share
- ✓ should schedule from 0 to 4 years (168ms)
- ✓ should schedule from 1 to 2 years (171ms)
- ✓ should schedule from 1 to 3 (160ms)
- ✓ should schedule from 0 to 1 (207ms)
- ✓ should schedule from 0 to now (200 days)
- ✓ should schedule from 0 to now (400 days) (231ms)
- ✓ should schedule from 0 to now (750 days) (248ms)
- ✓ should schedule from 200 to now (750 days) (246ms)
- ✓ should schedule from 200 to end (4 years) (156ms)
- ✓ should schedule from 200 to end (3 years) (168ms)
- ✓ should schedule from 400 to end (3 years) (147ms)
- ✓ should schedule from 400 to end of (3rd year) + 2 day (187ms)
- ✓ should stake on behalf of another user (948ms)
- ✓ should batch stake on bahalf of another users (480ms)
- ✓ should get user shares
- ✓ should get release time (451ms)
- ✓ should withdraw rewards after release time (513ms)
- ✓ should claim on behalf of another user (692ms)
- ✓ should batch claim on behalf of other users (2274ms)
- ✓ should get zero stream owner claimable amount if stream is inactive (243ms)
- ✓ should return aurora stream user shares if stream is zero (359ms)
- ✓ should return total amount of staked aurora (167ms)
- ✓ should get zero reward amount before stream start and stream end (296ms)

- ✓ should claim zero reward if stream did not start (733ms)
- ✓ should return if _before called twice whithin the same block
- ✓ should withdraw all rewards after release time (2229ms)
- ✓ should unstake all (417ms)
- ✓ should claim all rewards (678ms)
- ✓ should get reward per share for user (502ms)
- ✓ should calculate weighted shares (80ms)
- ✓ should get reward per share for a user (777ms)
- ✓ should get claimable amount (818ms)
- ✓ should restake the rest of aurora tokens (763ms)
- ✓ should return zero total aurora staked if touched equals zero
- ✓ should release rewards from stream start (530ms)
- ✓ should calculate stream claimable rewards from stream start (572ms)
- ✓ should claim rewards for a stream even if user staked before stream deployment (680ms)
- ✓ should be able to unstake and withdraw even if after the schedule ends (434ms)
- ✓ should only admin update the treasury address (526ms)
- ✓ should admin remove stream (551ms)
- ✓ should admin cancel stream proposal after expiry date (321ms)
- ✓ admin can claim streams on behalf of another user (1803ms)
- ✓ estimageGas staking with multiple streams (15992ms)
- ✓ estimageGas claiming all with multiple users (4219ms)
- ✓ should not return zero stake value when a user unstakeAll (1474ms)

- ✓ should user 1 stakes before user 2 but both stake very small but the same amount and unstake at the same time (1261ms)
- ✓ should both users get the same reward if they stake and unstake the same amount at the same time (1039ms)
- ✓ should user 2 should get double rewards if he has a double stake (1038ms)
- ✓ should return the right share calculations (1440ms)
- ✓ should return the right share calculations 2 (1322ms)
- ✓ should user 0 stake, then two new users stake and unstake the same amount at the same time (1347ms)
- ✓ should not have a possible race condition (1618ms)

Contract: Treasury

- ✓ should allow transfer ownership (75ms)
- ✓ should allow only owner pay rewards (83ms)
- ✓ should allow only manager add supported token (45ms)
- ✓ should allow only manager to remove supported token (107ms)
- ✓ should allow only manager to add a new manager (269ms)
- ✓ should allow only manager to remove a manager (257ms)
- ✓ should allow default admin role to withdraw some aurora funds (80ms)

Contract: JetStakingV1Upgrade

- ✓ should test JetStakingV1 change function signature (724ms)
- ✓ should test JetStakingV1 change in storage (1981ms)
- ✓ should test JetStakingV1 change in storage and logic (517ms)
- ✓ should test JetStakingV1 extra functionality (378ms)

Contract: TreasuryUpgrade

- ✓ should test Treasury change function signature (286ms)
- ✓ should test Treasury change in storage (283ms)
- ✓ hould test Treasury change in storage and logic (357ms)
- ✓ should test Treasury extra functionality (272ms)

86 passing (2m)

TEST COVERAGE RESULTS

BY AURORA TEAM

FILE	% STMTS	% BRANCH	% FUNCS
AdminControlled.sol	100	75	100
JetStakingV1.sol	99.31	67.31	100
Treasury.sol	100	50	100
All files	**99.37**	**66.86**	**100**

CODE COVERAGE AND TEST RESULTS FOR ALL FILES BY BLAIZE.SECURITY TEAM

Contract: AdminControlled

- ✓ Should set pause (173ms)
- ✓ Should not perform delegate call to self address (86ms)
- ✓ Should not perform delegate call to zero address (66ms)

Contract: JestStakingV1

- ✓ Should revert initialize if max weight < min weight (472ms)
- ✓ Should revert initialize if invalid address provided (366ms)
- ✓ Should revert initialize if schedule values are invalid (745ms)
- ✓ Should revert initialize if tau period equals 0 (282ms)
- ✓ Should revert initialize if schedule time is invalid (282ms)
- ✓ Should revert initialize if schedule reward is invalid (369ms)
- ✓ Should revert initialize if end reward is not zero (367ms)
- ✓ Should not propose stream if owner address is zero (476ms)
- ✓ Should not propose stream if reward token address is zero (366ms)
- ✓ Should not propose stream if max deposited amount is zero (257ms)
- ✓ Should not propose stream if stream expiration date is in the past (524ms)

- ✓ Should not propose stream if schedule values are invalid (355ms)
- ✓ Should not propose stream if tau value is zero (439ms)
- ✓ Should not propose stream if stream schedule times are invalid (646ms)
- ✓ Should not propose stream if stream schedule rewards are invalid (415ms)
- ✓ Should not propose stream if stream schedule end reward is not zero (507ms)
- ✓ Should not create stream if stream is not proposed (588ms)
- ✓ Should not let not owner of stream create (398ms)
- ✓ Should not let create stream more than once (996ms)
- ✓ Should not let create stream if proposal expired (572ms)
- ✓ Should not let create stream if reward amount > max deposit amount (589ms)
- ✓ Should not let remove aurora stream (100ms)
- ✓ Should not let remove not active stream (554ms)
- ✓ Should return zero claimable amount for owner if stream not active (555ms)
- ✓ Should not release aurora rewards if called not by stream owner (505ms)
- ✓ Should not release aurora rewards if stream not active (535ms)
- ✓ Should not let change treasury address to zero (92ms)
- ✓ Should not stake on behalf of users if arrays length mismatch (197ms)
- ✓ Should not stake on behalf of users if batch amount is invalid (562ms)
- ✓ Should claim on behalf of another user (1419ms)
- ✓ Should batch claim on behalf of other users (1158ms)

- ✓ Should not let withdraw rewards before release time (1378ms)
- ✓ Should not batch withdraw before release time (1878ms)
- ✓ Should not move rewards to pending if stream is not active (400ms)
- ✓ Should return aurora shares if aurora stream provided (247ms)
- ✓ Should return 0 reward amount if stream not started yet or finished (795ms)
- ✓ Should not return latest RPS for aurora stream id (77ms)
- ✓ Should not return latest RPS if total stream shares is 0 (41ms)
- ✓ Should return total amount of staked aurora (390ms)
- ✓ Should not get start and end index if wrong parameters (682ms)
- ✓ Should not return reward schedule start < schedule beginning (703ms)
- ✓ Should not return reward schedule end > schedule finish (777ms)
- ✓ Should not move rewards to pending if aurora stream id provided (953ms)
- ✓ Should not move rewards to pending if stream not started yet (1012ms)
- ✓ Should not move rewards to pending for not active stream (1234ms)
- ✓ Should not let unstake 0 (427ms)
- ✓ Should not let unstake more than stake balance (701ms)
- ✓ Should revert unstaking if user shares are zero (291ms)
- ✓ Should revert initialise if treasury address is zero (260ms)
- ✓ Should revert initialise if unsupported token provided (283ms)

- ✓ Should revert proposing stream if min deposit amount is zero (193ms)
- ✓ Should revert proposing stream if min deposit amount > max deposit amount (176ms)
- ✓ Should revert proposing stream if max deposit amount not equal rewardSchedule[0] (202ms)
- ✓ Should revert proposing stream if unsupported token provided (221ms)
- ✓ Should revert canceling stream if stream is not in proposed state (65ms)
- ✓ Should revert creating stream if reward amount < min deposit amount (381ms)
- ✓ Should not release rewards for owner for stream 0 (52ms)
- ✓ Should return zero rewards if last update > stream end (619ms)

Contract: Treasury

- ✓ Should revert initialize if provided token is zero (185ms)
- ✓ Should not add already supported token (81ms)
- ✓ Should not remove unsupported token (114ms)
- ✓ Should not pay reward in unsupported token (101ms)

Contract: User flow

- ✓ Creating stream 1 (650ms)
- ✓ Creating stream 2 (537ms)
- ✓ Creating stream 3 (574ms)
- ✓ User1 stakes AURORA tokens (316ms)
- ✓ User2 stakes AURORA tokens (982ms)
- ✓ User1 and User2 move rewards to pending for stream 1 (1202ms)
- ✓ User1 and User2 move rewards to pending for stream 2 (929ms)
- ✓ User1 and User2 move rewards to pending for stream 3 (1258ms)
- ✓ Waiting release period for streams 1, 2, 3

- ✓ User1 and user2 withdraw rewards for stream 1 (444ms)
- ✓ User1 and user2 withdraw rewards for stream 2 (453ms)
- ✓ User1 and user2 withdraw rewards for stream 3 (436ms)
- ✓ User1 and user2 unstake AURORA (1510ms)
- ✓ Waiting release period for AURORA stream
- ✓ User1 and user2 withdraw rewards for AURORA stream (473ms)

166 passing(3m)

TEST COVERAGE RESULTS

BY BLAIZE.SECURITY TEAM

FILE	% STMTS	% BRANCH	% FUNCS
AdminControlled.sol	100	100	100
JetStakingV1.sol	100	97.44	100
Treasury.sol	100	100	100
All files	**99.45**	**95.56**	**98.78**

DISCLAIMER

The information presented in this report is an intellectual property of the customer including all presented documentation, code databases, labels, titles, ways of usage as well as the information about potential vulnerabilities and methods of their exploitation. This audit report does not give any warranties on the absolute security of the code. Blaize.Security is not responsible for how you use this product and does not constitute any investment advice.

Blaize.Security does not provide any warranty that the working product will be compatible with any software, system, protocol or service and operate without interruption. We do not claim the investigated product is able to meet your or anyone else requirements and be fully secure, complete, accurate and free of any errors and code inconsistency.

We are not responsible for all subsequent changes, deletions and relocations of the code within the contracts that are the subjects of this report.

You should perceive Blaize.Security as a tool which helps to investigate and detect the weaknesses and vulnerable parts that may accelerate the technology improvements and faster error elimination.