HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Aurora Labs
Date: June 8th, 2022

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for Aurora Labs
Approved By	Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type	Staking
Platform	EVM
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Website	https://aurora.dev/
Timeline	25.04.2022 - 07.06.2022
Changelog	02.05.2022 - Initial Review 17.05.2022 - Second Review 08.06.2022 - Third Review

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by Aurora Labs (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

Scope

The scope of the project is smart contracts in the repository:

Initial review scope

Repository: https://github.com/aurora-is-near/aurora-staking-contracts
Commit: b59cc2926cbaf31d351c9d048b5a2dc07cbddf70
JS tests: Yes

Technical Documentation:

Type: Whitepaper (some functional requirements included)
Link: https://forum.aurora.dev/t/aurora-staking-and-the-community-treasury/75

Type: Technical description
Link: https://github.com/aurora-is-near/aurora-staking-contracts/blob/main/README.md

Type: Functional requirements
Link: https://github.com/aurora-is-near/aurora-staking-contracts/blob/main/docs/README.md

Contracts:

File: ./contracts/AdminControlled.sol
SHA3: b9a609c71670f30e7885b574c9a78475bf1947d1a0f7afb200f7005a

File: ./contracts/DelegateCallGuard.sol
SHA3: f285c7562db714aaeaab6411ba836a30fab0a25f8bdd22b05bba515b

File: ./contracts/ITreasury.sol
SHA3: 09194dbef79523270261ca99fe89800597ef499b3413c53888c8a58f

File: ./contracts/JetStakingV1.sol
SHA3: c3b6f961b4283c6d08b65d3ada9eb2fe60bcf15240314b8f735719f5

File: ./contracts/Treasury.sol
SHA3: f3ac3f67dacffb90e47c197919b313a3ab81eee1fb77a0e64bf3d23d

Second review scope

Repository: https://github.com/aurora-is-near/aurora-staking-contracts
Commit: dc0a4893e8017a13e3c5a735eca1913c467acac6
JS tests: Yes

Technical Documentation:

Type: Whitepaper (some functional requirements included)
Links:
https://forum.aurora.dev/t/aurora-staking-and-the-community-treasury/75
https://forum.aurora.dev/t/aurora-staking-v2/243
https://forum.aurora.dev/t/setting-up-the-aurora-staking/254

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Type: Technical description
Link: https://github.com/aurora-is-near/aurora-staking-contracts/blob/main/README.md

Type: Functional requirements
Link: https://github.com/aurora-is-near/aurora-staking-contracts/blob/main/docs/README.md

Contracts:
File: ./contracts/AdminControlled.sol
SHA3: d1ea0625e760f4237f518942b18e2a425bbf0a5e3cad1e8d51872498

File: ./contracts/ITreasury.sol
SHA3: aa9e381645142126780ab317bba703f577b3c60c4b9af28bb557d655

File: ./contracts/JetStakingV1.sol
SHA3: b242d63bf4123ad0f6f8d17c7b21294ade1f72b979133777f331a41c

File: ./contracts/Treasury.sol
SHA3: 853aef9b54c385858035d85b711656fc11a48e0eb9a4a232573e3829

Third review scope

Repository: https://github.com/aurora-is-near/aurora-staking-contracts
Commit: e32dc4197bd3cb4db4178695e969f58b053821b3
JS tests: Yes

Technical Documentation:
Type: Whitepaper (some functional requirements included)
Links:
https://forum.aurora.dev/t/aurora-staking-and-the-community-treasury/75
https://forum.aurora.dev/t/aurora-staking-v2/243
https://forum.aurora.dev/t/setting-up-the-aurora-staking/254

Type: Technical description
Link: https://github.com/aurora-is-near/aurora-staking-contracts/blob/main/README.md

Type: Functional requirements
Link: https://github.com/aurora-is-near/aurora-staking-contracts/blob/main/docs/README.md

Contracts:
File: ./contracts/AdminControlled.sol
SHA3: 5c6029ac553c21db56465e2b88124ab6cf1726652643a9a1b1dc8530

File: ./contracts/ITreasury.sol
SHA3: aa9e381645142126780ab317bba703f577b3c60c4b9af28bb557d655

File: ./contracts/JetStakingV1.sol
SHA3: 901b391dcdf3a0324da90a9f61e58d0ad6a9f05f63d361bbe58cb383

File: ./contracts/Treasury.sol
SHA3: 493089a0726cf097703af2bf9cf427bba32859efd0a2e10abf9d54a8

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions.
Medium	Medium-level vulnerabilities are important to fix; however, they cannot lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Executive Summary

The score measurement details can be found in the corresponding section of the methodology.

Documentation quality

The Customer provided a good technical description and functional requirements. The total Documentation Quality score is **10** out of **10**.

Code quality

The total CodeQuality score is **8** out of **10**. Code refers to the UI part that is out of scope to be clear enough to prevent human factors.

Architecture quality

The architecture quality score is **10** out of **10**. The logic is carefully separated by files, and each part has its purpose.

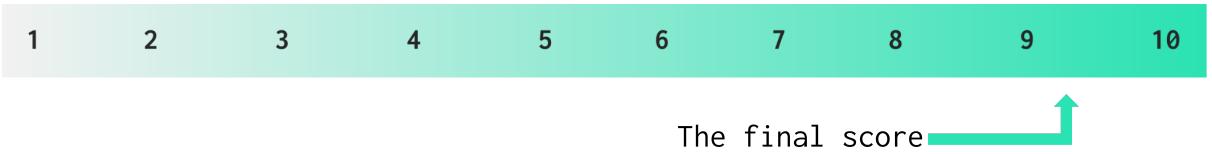
Security score

As a result of the audit, the code contains **1** medium severity issues. The security score is **9** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **9.1**



Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Checked Items

We have audited provided smart contracts for commonly known and more specific vulnerabilities. Here are some of the items that are considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Not Relevant
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Passed
Access Control & Authorization	CWE-284	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	SWC-106	The contract should not be destroyed until it has funds belonging to users.	Not Relevant
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Uninitialized Storage Pointer	SWC-109	Storage type should be set explicitly if the compiler version is < 0.5.0.	Not Relevant
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Not Relevant
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Passed
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless it is required.	Passed
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Failed

Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Passed
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Passed
Signature Unique Id	SWC-117 SWC-121 SWC-122	Signed messages should always have a unique id. A transaction hash should not be used as a unique id.	Passed
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes.	Passed
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Passed
Presence of unused variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Passed
EIP standards violation	EIP	EIP standards should not be violated.	Not Relevant
Assets integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions.	Failed
User Balances manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Failed
Data Consistency	**Custom**	Smart contract data should be consistent all over the data flow.	Passed
Flashloan Attack	**Custom**	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant
Token Supply manipulation	**Custom**	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the customer.	Not Relevant
Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block gas limit.	Failed

Style guide violation	**Custom**	Style guides and best practices should be followed.	Passed
Requirements Compliance	**Custom**	The code should be compliant with the requirements provided by the Customer.	Passed
Repository Consistency	**Custom**	The repository should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Tests Coverage	**Custom**	The code should be covered with unit tests. Test coverage should be 100%, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

System Overview

AURORA Staking is a system of contracts that allows to stake the Aurora token and get other tokens as rewards. The system consists of the following parts:

- *AdminControlled* - admin panel for contract owner to fix unexpected issues (is implemented both for *Treasury and JetStakingV1*).
- *Treasury* + *ITreasury* - treasury for tokens, all unpaid rewards are stored there.
- *JetStakingV1* - staking contract that manages streams of rewards and implements base staking logic for users.

Privileged roles

- Owners of *DEFAULT_ADMIN_ROLE* role can do:
  - any changes in contracts' storages
  - use delegatecalls to untrusted sources
  - transfer any funds from *Treasury*
  - change *Treasury* address used in *JetStakingV1*
  - unpause contract
- Owners of *PAUSE_ROLE* role can pause contract:
  - paying rewards from *Treasury*
  - creating reward streams
  - releasing rewards to the stream owner
  - staking, unstaking, and withdrawing rewards to users
  - airdropping the Aurora token
- Owners of *TREASURY_MANAGER_ROLE* role can:
  - update list of supported tokens in *Treasury*
- Owners of *AIRDROP_ROLE* role can:
  - create stakings for other users and in such a way airdropping the Aurora token
- Owners of *CLAIM_ROLE* role can:
  - force move rewards to pending for specified users and streams
- Owners of *STREAM_MANAGER_ROLE* role can:
  - manage staking reward streams (propose, cancel proposal, remove)

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Findings

Critical

No critical severity issues were found.

High

1. Wrong user shares calculation

According to documentation, the user shares value should be rounded up, but in the code, just 1 is added. To round, the result of the division should be checked that rounding is needed (result * denominator < numerator).

Adding additional value to user shares may lead to wrong contract behavior. Example (zero rewards provided for the period):

- transaction_1..2: 2 users stake 1 Aurora token
- result: users get different share values:
  - user_1 -> 1
  - user_2 -> 2
- transaction_3: trying unstake for user_1 (*unstakeAll*)
- result: *totalAmountOfStakedAurora* < *totalAuroraShares* => calculated *stakeValue = 0* => unstaking 0 amount = did user lose funds?
- transaction_4: trying unstake for user_2 (*unstakeAll*)
- transaction_5: trying unstake for user_1 (*unstakeAll*)
- result: transaction_4 and transaction_5 are successful

The success of unstaking depends on the stake status of another user.

This could lead to locking user assets (or part of them) on contract without the ability to withdraw them.

Contract: JetStakingV1

Function: _stake, unstakeAll, unstake

Recommendation: check the logic, update unstake functions to consider additional added value or remove the addition from stake function, implement unit tests that cover cases of staking by multiple users, implement documentation for the internal staking algorithm.

Status: Fixed (second review)

2. Possible broken calculations

Calculations of reward during the *endIndex* period use *startIndex* period. If the *endIndex* period is bigger than the *startIndex* period, more rewards would be distributed than allocated.

The function is overwhelmed with template calculations.

This could lead to wrong reward calculation and possible double spending.

Contract: JetStakingV1

Function: rewardsSchedule

Recommendation: reduce code duplications, use the right period for reward calculations.

Status: Fixed (second review)

3. **Possible race conditions**

The withdrawal amount and shares amount highly depend on the order of transactions. Example (all happens in one block):

- transaction_1: user_1 stakes 1 Aurora
- transaction_2: user_2 stakes 10**18 Aurora
- transaction_3..4: both users unstakes all
- result: both users get a significant amount in pending as they have mostly equal shares amount, but if transaction_1 and transaction_2 are swapped, user_1 will receive a really small withdrawal amount and user_2 the rest because shares amount values are different

It is more profitable for user_1 to stake before user_2, but for user_2 it is better to be the first too, so race conditions appear.

Contract: JetStakingV1

Functions: _stake, _unstake, _before

Recommendation: check calculating stream shares logic, implement corresponding documentation for users to understand how many tokens they will get on withdrawal, cover this case with unit tests.

Status: Mitigated (second review). Such a situation is not possible with a large number of users.

4. **Missing ability to pause contract**

According to the documentation, owners of *PAUSE_ROLE* should be able to pause contracts, but this ability is missed.

This could lead to the helplessness of owners of *PAUSE_ROLE* in critical situations.

Contract: AdminControlled

Function: adminPause

Recommendation: review the functionality and fix it according to the documentation.

Status: Fixed (third review)

5. **Distributing Aurora tokens**

According to documentation, the Aurora stream is compounded, so no Aurora should be distributed from the contract.

Due to wrong shares calculation, Aurora is distributed from the contract. Example (all happens in one block):

- transaction_1: user stakes 10**18 Aurora
- transaction_2: user unstakes all
- result: user gets more pending amount than deposited

The user may repeat the action every *tau* period to get Aurora tokens from *Treasury*.

Contract: JetStakingV1

Functions: _before

Recommendation: reimplement calculating stream shares logic, implement documentation to understand how assets are distributed through users, cover this case with unit tests.

Status: Fixed (third review)

Medium

1. **Missing managing roles**

   To prevent the significant impact of the previous admin, it is better to revoke all actual roles on ownership transfer.

   Contracts: AdminControlled, JetStakingV1, Treasury

   Function: transferOwnership

   Recommendation: implement *transferOwnership* function in mentioned contracts, revoke all actual roles from the old admin and grant them to the new one.

   Status: Fixed (second review)

2. **Missing setting admin account**

   To prevent sending assets to zero account, it is better to set the admin account in the initialize function.

   Contract: AdminControlled

   Function: __AdminControlled_init

   Recommendation: initialize *admin* with *msg.sender*.

   Status: Fixed (second review)

3. **Main stream could be canceled**

   Aurora stream is set to be proposed on contract creation, it could be canceled, and assets needed for other proposed streams could be withdrawn from the staking contract.

   This could lead to possible double spending.

   Contract: JetStakingV1

   Functions: initialize, cancelStreamProposal

Recommendation: set Aurora stream to be not proposed or check this case in the cancellation function.

Status: Fixed (second review)

4. **Missing validations**

   According to the documentation, before running the *initialize* function of *JetStakingV1*, the Aurora token should be added to the list of supported tokens, and the balance of *Treasury* should be big enough.

   According to the documentation, *updateTreasury* function should be run only when the contract is paused.

   Mentioned validation should be implemented to prevent human factors.

   Contract: JetStakingV1

   Functions: initialize, updateTreasury, stake, unstake, unstakeAll, stakeOnBehalfOfOtherUsers, stakeOnBehalfOfAnotherUser

   Recommendation: implement these checks.

   Status: Fixed (second review)

5. **Mixing role purposes**

   DEFAULT_ADMIN_ROLE is used in *AdminControlled* contract for dangerous low-level operations and in *Treasury* contract for dangerous manual reward paying, but it is used in *JetStakingV1* contract for managing streams. It is a lighter task and could be assigned to a limited liability account, so it is better to have another role managing it.

   Contract: JetStakingV1

   Recommendation: implement a role for managing reward streams.

   Status: Fixed (second review)

6. **Locking tokens in Treasury**

   Creating stream proposals does not add selected tokens to the list of supported in the *Treasury* contract. Rewards may not be accessible after stream creation.

   Contract: JetStakingV1

   Function: proposeStream

   Recommendation: check if the proposed stream token is included in the list of supported tokens or add it automatically.

   Status: Fixed (second review)

7. **Transfer can fail**

   Transfer can fail if the destination address is a contract with a fallback function.

Contract: AdminControlled

Function: adminSendEth

Recommendation: send Ether via call and pass Gas limit via function parameter.

Status: Fixed (second review)

8. **Missing emitting events**

   TokenAdded event should be emitted on adding token contract address to list of supported, but it is missed in on contract creation.

   Contract: Treasury

   Function: initialize

   Recommendation: emit an event every time the mapping is updated.

   Status: Fixed (third review)

9. **Code with no effects**

   According to documentation, Aurora main stream is not needed anymore, so logic (including checks, additional structure fields etc.) linked with it should be removed from the contract code. Unneeded code takes additional Gas and makes development harder.

   Contract: JetStakingV1

   Recommendation: remove initializing of the stream and corresponding checks.

   Status: Mitigated (second review). The code is required if a new instance of the contract is deployed.

10. **Missing functionality**

    According to the documentation, *stake*, *unstake*, *unstakeAll*, *stakeOnBehalfOfOtherUsers*, *stakeOnBehalfOfAnotherUser* functions should claim rewards if the selected user has actual staking.

    Mentioned functionality should be implemented to prevent human factors.

    Contract: JetStakingV1

    Functions: initialize, updateTreasury, stake, unstake, unstakeAll, stakeOnBehalfOfOtherUsers, stakeOnBehalfOfAnotherUser

    Recommendation: implement the declared functionality.

    Status: Reported

11. **Possible gas exceeding**

    The amount of streams only increases and in such a way functions that cycling each stream may fail according to the block Gas limit. Most of all, it affects the functions which do not have alternatives without a cycle.

Contract: JetStakingV1

Function: _stake, _before

Recommendation: implement an ability to remove old streams and keep their quantity under the provided number for which Gas limit is not exceeded.

Status: Mitigated. The Customer confirmed that number of streams will not grow too much.

Low

1. Wrong constant value

According to the documentation, the constant *FOUR_YEARS* should contain 4 years in seconds value. According to the Gregorian calendar, the value is calculated as 60 seconds * 60 minutes * 24 hours * 365.2425 days * 4 years = 126227808 seconds, but it is set to 126227704 in the code.

Contract: JetStakingV1

Recommendation: set the constant to the right calculated value or provide comments about the purpose of making the value different.

Status: Fixed (second review)

2. Public function that could be declared external

To save Gas, *public* functions that are never called by the contract should be declared *external*.

Contract: JetStakingV1

Function: getStreamSchedule

Recommendation: use the *external* attribute for functions never called from the contract.

Status: Fixed (second review)

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed by the best industry practices at the date of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The audit makes no statements or warranties on the security of the code. It also cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the audit cannot guarantee the explicit security of the audited smart contracts.