HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Aurora labs
Date: January, 18th, 2023

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 Audits Department Head at Hacken OU
Type	DAO Council Election and Vote Token
Platform	EVM
Network	Ethereum
Language	Solidity
Methodology	Link
Website	https://aurora.dev/
Timeline	10.10.2022 - 11.01.2023
Changelog	13.10.2022 - Initial Review 14.11.2022 - Second Review 18.01.2023 - 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 review and security analysis of smart contracts in the repository:

Initial review scope

Repository:
https://github.com/aurora-is-near/aurora-voting-contracts
Commit:
98de1af2100d9b0e097105c1a466c47867a68a68

Repository:
https://github.com/aurora-is-near/aurora-election-contracts
Commit:
a46703998dff34fa76861336eb648019bb118cbf

Documentation:
Technical description
Technical description
Functional requirements
Contracts:
File: ./contracts/AuroraVoteTokenV1.sol
SHA3: 0f71b2b022a77578e63b440007b1bd348c7cce013c03dd693a501dcd39019740

File: ./contracts/ERC20VotesUpgradeable.sol
SHA3: e3915407fdb64bc520768c923f67484de509fe954fc3a346a0902a24e8fa4561

File: ./contracts/AdminControlled.sol
SHA3: 6a3bb8b4f5eef3e1e7e0a86be0f941ba19f66a8d7b3dff03bbf6f69f432f20b3

File: ./contracts/ElectionManager.sol
SHA3: 10cea2ee89ed7c901c35c50cc2916be89ce79400418e6da0369858a2ac1f927b

Second review scope

Repository:
https://github.com/aurora-is-near/aurora-voting-contracts
Commit:
9abea3884426bd23c0bb4a9a253799790356fb2b

Repository:
https://github.com/aurora-is-near/aurora-election-contracts
Commit:
a93d31ca487bbb78bb70f9cf8e0df8ae9fa5777d

Documentation:
Technical description
Technical description
Functional requirements
Contracts:

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

```
File: ./contracts/AuroraVoteTokenV1.sol
SHA3: ad4807aef0d8500bbc2818de3df13eb1a0bc28aab6ef5d1da56bbf7eebbb9c7a

File: ./contracts/ERC20VotesUpgradeable.sol
SHA3: 767697eb21c4f60a7d9023aa078ad0ebe31596687b87bd9406e6f74d6dbffd4a

File: ./contracts/AdminControlled.sol
SHA3: e540f21870f7e1e2096c47abd83295ce646aacd80c621c2f63076732648cf81f

File: ./contracts/ElectionManager.sol
SHA3: 8e8211655409baf44aa644a63b87e5aa1e6672ba07a42414e3cf982e24a1bb4e
```

Third review scope

Repository:
https://github.com/aurora-is-near/aurora-voting-contracts
Commit:
3c9cc04dc33edc3cd2ab0491632a79a6d4dfef49

Repository:
https://github.com/aurora-is-near/aurora-election-contracts
Commit:
b530a126aebf1220bb0da29607160ac6b23556d2

Documentation:
Technical description
Technical description
Functional requirements
Contracts:

```
File: ./contracts/AuroraVoteTokenV1.sol
SHA3: b18bec78a463378926e6d134136ee510d18e32d943fa051db64d7dded272337e

File: ./contracts/ERC20VotesUpgradeable.sol
SHA3: aac38beb5d4600c12b2c03093d458fc529d33bf2de26f21085f57603257e1663

File: ./contracts/AdminControlled.sol
SHA3: 8e9f3b6482242aab33735b962803e19b3daf9d900f6e2ed0ff40102abce1c10c

File: ./contracts/ElectionManager.sol
SHA3: fb92ef4e1385ea4006f7feb9103e131a41b7223af6b5663a8b4520470de71055
```

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 scoring methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**.

- Functional requirements are good.
- Technical description is good.

Code quality

The total Code Quality score is **9** out of **10**.

- The code follows best practices and style guidelines.
- The development environment is configured.
- Low severity issues reduce code quality.

Test coverage

Test coverage of the project is **100%**.

- Code is fully covered by unit tests.
- Only 0x addresses that cannot be tested are omitted..

Security score

As a result of the audit, the code contains **2** low severity issues. The security score is **10** 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: **9.8**.

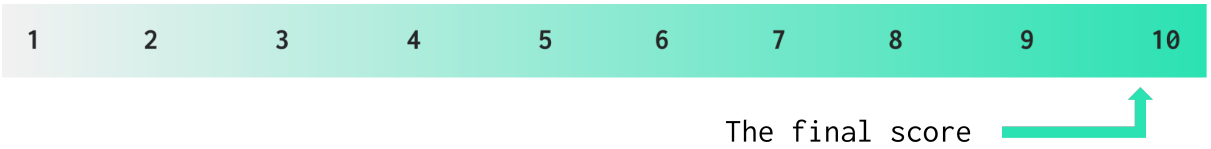

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
13 October 2022	5	3	3	0
10 November 2022	1	1	1	0
11 January 2023	2	0	0	0

Checked Items

We have audited the Customers' smart contracts for commonly known and more specific vulnerabilities. Here are some items 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.	Passed
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 self-destructible while 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
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Failed
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 required.	Passed
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed

Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Not Relevant
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 EIP-155	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery	Not Relevant
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 or be predictable.	Not Relevant
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.	Passed
Assets integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions.	Passed
User Balances manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
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.	Passed

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.	Passed
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
Environment Consistency	**Custom**	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Secure Oracles Usage	**Custom**	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant
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
Stable Imports	**Custom**	The code should not reference draft contracts, that may be changed in the future.	Passed

System Overview

Aurora Labs is a DAO council members election system with the following contracts:

- *AuroraVoteTokenV1* — a custom ERC-20 token that mints all initial supply at initialization. Additional minting is not allowed. It does not allow users to use the *transfer* and *transferFrom* functions but instead offers them the possibility to delegate tokens using the *delegate* function.
- *ERC20VotesUpgradeable* - a custom ERC-20 token contract inherited by the former.
- *ElectionManager* — a contract that handles the election process.
- *AdminControlled* — a contract that sets admin privileges, inherited by the former.

Privileged roles

- AuroraVoteTokenV1 roles:
  - DEFAULT_ADMIN_ROLE - allowed to do admin operations such as grant roles, upgrade contract.
  - WHITELISTED_TRANSFER_ROLE - allowed to transfer tokens
  - WHITELISTED_TRANSFER_FROM_ROLE - allowed to transfer tokens from a chosen address.
- ElectionManager roles:
  - ELECTION_MANAGER - allowed to create election,stop election, add candidates application, remove candidates application, update candidates application, update election application intervals, update election intervals.
  - DEFAULT_ADMIN_ROLE - allowed to do admin operations like grant roles, upgrade contract, change storage values.
  - PAUSE_ROLE - allowed to perform admin pause.

Risks

- **The entirety of the system is controlled by an out-of-scope, centralized off-chain system.** This includes:
  - The entirety of the **VOTE token supply**, as well as the **distribution of VOTE token rewards**
  - **Most election functionalities**, barring the storage of the results
  - **Burning** of the **remaining unstreamed VOTE** tokens, and general **transfers** of the VOTE token. Performed by the stream admin.
- Only addresses with privileges can use the transferFrom function of the VOTE token. Therefore, the ElectionManager contract must be whitelisted in the token contract in order to process the user's vote.

- Some functionality from the documentation is implemented in out-of-scope contracts, and its validity cannot be verified.
- In ElectionManager.sol, the functions getCurrentElectionResults and getElection loop over a dynamic array.This array size is determined by the number of candidates in an election. If too many candidates meet the requirements declared, this can lead to an Out-of-Gas exception if the off-chain computation tracking does not work properly.

Findings

■■■■ Critical

No critical severity issues were found.

■■■ High

1. **Requirements Violation**

   The documentation states that the contract has a burn feature, but the code does not implement it.

   According to the provided documentation, “The staking stream will be stopped in advance before the actual election of the Council Members. All remaining Vote Tokens will be burned.”

   However, there is no mechanism to burn tokens based on the staking stream state.

   Path: ./contracts/AuroraVoteTokenV1.sol

   Recommendation: align the implementation with the requirements.

   Status: Mitigated (the stream admin gets the remaining un-streamed VOTE then burns them manually)

2. **Highly Permissive Role Access**

   The default admin of the election contract can update storage values at any time.

   Any storage data can be changed.

   Path: ./contracts/AdminControlled.sol : function adminSstore(), adminSstoreWithMask()

   Recommendation: remove the mentioned functionality from the contract.

   Status: Fixed (a93d31ca487bbb78bb70f9cf8e0df8ae9fa5777d)

■■ Medium

1. **Missing Events Emit on Changing Important Values**

   It is recommended to emit events after changing important values. This will make it easy for everyone to notice such changes.

   Paths: **./**contracts/ElectionManager.sol : function stopElection(), updateElectionApplicationIntervals(), updateElectionIntervals(), vote()

   ./contracts/AdminControlled.sol : adminPause(), adminSstore(), adminSstoreWithMask()

   Recommendation: implement event emits after changing contract values.

   Status: Fixed (a93d31ca487bbb78bb70f9cf8e0df8ae9fa5777d)

2. Potential Out-of-Gas Exception

In ElectionManager.sol, the functions getCurrentElectionResults and getElection loop over a dynamic array.

This array size is determined by the number of candidates in an election. If too many candidates meet the requirements declared, this can lead to an Out-of-as exception.

Path: **./**contracts/ElectionManager.sol : function getCurrentElectionResults(), getElection

Recommendation: limit the number of candidates that can participate in the election or make sure these functions are only called off-chain, where Gas limits are not an issue.

Status: Mitigated (The limit here is unnecessary as the election result will rely on off-chain computation by tracking the events.)

Low

1. Misleading Function

The *delegate* function executes the exact same code as the *transfer* function. The function name contradicts its functionality.

Path: ./contracts/AuroraVoteTokenV1.sol

Recommendation: use *transfer* function only to override the ERC20 transfer function and remove the *delegate* function.

Status: Mitigated (It is part of the business requirements, but it was verbally communicated. The intention is to provide both interfaces (transfer and delegate). The transfer is only for whitelisted addresses, however anyone can call the delegate function.)

2. Floating Pragma

The project uses floating pragmas ^0.8.0.

Path: **./**contracts/ERC20VotesUpgradeable.sol

Recommendation: consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment.

Status: Fixed (9abea3884426bd23c0bb4a9a253799790356fb2b)

3. Functions that Can Be Declared External

In order to save Gas, public functions that are never called in the contract should be declared as external.

Path: **./**contracts/AuroraVoteTokenV1.sol : function delegate()

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

Status: Fixed (9abea3884426bd23c0bb4a9a253799790356fb2b)

4. **Style Guide Violation**

   Within the external functions, the *view* should be last.

   Path: ./contracts/ElectionManager.sol

   Recommendation: follow the official Solidity guidelines.

   Status: Fixed (a93d31ca487bbb78bb70f9cf8e0df8ae9fa5777d)

5. **Variable Shadowing**

   Function parameters _name, _symbol shadows existing variables from ERC20VotesUpgradeable inherited contract.

   Path: ./contracts/AuroraVoteTokenV1.sol

   Recommendation: rename related arguments.

   Status: Fixed (9abea3884426bd23c0bb4a9a253799790356fb2b)

6. **Assert Violation**

   Properly functioning code should never reach a failing assert statement.

   Path: ./contracts/ ElectionManager.sol : function getElectionCandidateUserVoteCount(), getCandidateByIndex()

   Remediation: this has been fixed for the following functions: getTotalElectionVotes(), getCandidateByAddress(), getElection()

   Recommendation: if the exception is indeed caused by unexpected behavior of the code, fix the underlying bugs that allow the assertion to be violated.

   Status: Reported

7. **Missing Zero Address Validation**

   Address parameters are being used without checking against the possibility of 0x0.

   This can lead to unwanted external calls to 0x0.

   Path: ./contracts/ElectionManager.sol : function initialize(), _validateAndInitializeElection(), addCandidateApplication()

   ./contracts/AdminControlled.sol : function adminDelegatecall()

   Recommendation: implement zero address checks.

   Status: New

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on the best industry practices at the time 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 report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted to and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and 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.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

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, Consultant cannot guarantee the explicit security of the audited smart contracts.