QuillAudits

AUDIT REPORT

December, 2024

For

ALVARA

Table of Content

Executive Summary 03

Number of Severity per issues 04

Checked Vulnerabilities 05

Techniques & Methods 07

Types of Severity 09

Types of Severity 10

High Severity Issues 11

1. _disableInitializers() should be called in the constructor 11

2. increaseAmount() function allows to increase amount in inactive pools as well 12

Low Severity Issues 13

1. Small increase amount does not affect voting power 13

2. Improper token burn function used in increaseAmount() and _stake() 14

3. Use CEI pattern 15

Informational Severity Issues 16

1. Add check for rewardID is present 16

2. Gas Optimisations 16

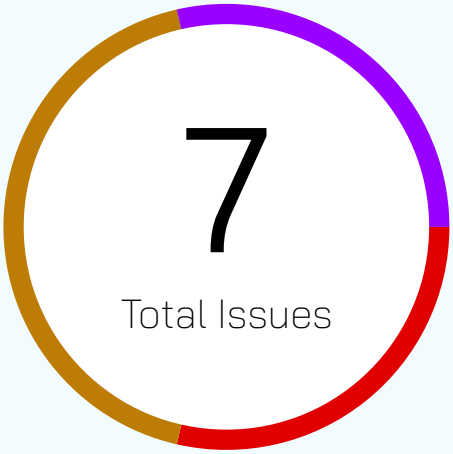
Functional Tests 18

Closing Summary & Disclaimer 20

Executive Summary

Project name	Alvara - Staking Contract
Overview	The Alvara staking contract is used to stake Alavara tokens in return user gets veAlva tokens which can be used for DAO voting in the Alvara ecosystem.
Project URL	https://alvara.xyz/
Audit Scope	https://github.com/Alvara-Protocol/alvara-contracts/tree/feat/staking1/contracts Branch: feat/staking1
Contracts under scope	contracts/AlvaStakeProtocol.sol contracts/tokens/veAlva.sol
Language	Solidity
Blockchain	EVM
Methods	Manual Analysis, Functional Testing, Automated Testing
Review 1	6th December 2024 - 13th December 2024
Updated Code Received	18th December 2024
Review 2	18th December 2024 - 20th December 2024
Fixed in	95a4236d188aee4f901b2b4e674c7f2e7699ca0e

Number of Issues per Severity



High	2 (28.57%)
Medium	3 (42.86%)
Low	0 (0.00%)
Informational	2 (28.57%)

Severity

Issues	High	Medium	Low	Informational
Open	0	0	0	0
Resolved	**2**	**2**	0	0
Acknowledged	0	**1**	0	**2**
Partially Resolved	0	0	0	0

Checked Vulnerabilities

- ✓ Re-entrancy
- ✓ Timestamp Dependence
- ✓ Gas Limit and Loops
- ✓ DoS with Block Gas Limit
- ✓ Transaction-Ordering Dependence
- ✓ Use of tx.origin
- ✓ Exception Disorder
- ✓ Gasless Send
- ✓ Balance Equality
- ✓ Byte Array
- ✓ Transfer Forwards All Gas
- ✓ ERC20 API Violation
- ✓ Compiler Version Not Fixed
- ✓ Redundant Fallback Function
- ✓ Send Instead of Transfer
- ✓ Style Guide Violation
- ✓ Unchecked External Call
- ✓ Unchecked Math
- ✓ Unsafe Type Inference
- ✓ Implicit Visibility Level
- ✓ Access Management
- ✓ Arbitrary Write to Storage
- ✓ Centralization of Control
- ✓ Ether Theft
- ✓ Improper or Missing Events
- ✓ Logical Issues and Flaws
- ✓ Arithmetic Computations Correctness
- ✓ Race Conditions/Front Running
- ✓ Malicious Libraries
- ✓ SWC Registry
- ✓ Address Hardcoded
- ✓ Divide Before Multiply

- [x] Integer Overflow/Underflow
- [x] Dangerous Strict Equalities
- [x] Tautology or Contradiction
- [x] Return Values of Low-Level Calls
- [x] Missing Zero Address Validation
- [x] Private Modifier
- [x] Revert/Require Functions
- [x] Multiple Sends
- [x] Upgradeable Safety
- [x] Using Delegatecall
- [x] Using Inline Assembly
- [x] Using Throw

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 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

Remix IDE, Foundry, Solhint, Mythril, Slither, Solidity statistical analysis.

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 Issues

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

Security vulnerabilities identified that must be resolved and are currently unresolved.

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

_disableInitializers() should be called in the constructor

Resolved

Path

contracts/tokens/veAlva.sol & contracts/AlvaStakeProtocol.sol

Description

When using proxy patterns for upgradeable contracts, the initializer modifier in OpenZeppelin's upgradeable contracts ensures that initialization logic is executed only once. However, constructors in upgradeable contracts can still inadvertently allow reinitialization through malicious calls if not explicitly disabled. OpenZeppelin provides the disableInitializers() function to prevent the initialization function from being called on the implementation contract itself.

If _disableInitializers() is not invoked in the constructor, it leaves the implementation contract vulnerable to accidental or malicious reinitialization, which could compromise contract security.

Recommendation

Add a call to _disableInitializers() in the constructor of the contract.

Impact

Malicious actors could exploit the absence of disableInitializers to reinitialize the contract and gain unauthorized access to sensitive data or functions.

increaseAmount() function allows to increase amount in inactive pools as well

Resolved

Path

contracts/AlvaStakeProtocol.sol

Function

increaseAmount()

Description

The increaseAmount function does not validate the status of the pool before processing the request. This allows users to increase their staked amount even if the pool is inactive or unavailable for staking. The absence of a pool status check can lead to unintended behaviors, such as locking funds in inactive or deprecated pools.

Recommendation

Add a pool status check to ensure that the pool is active before allowing the staking amount to increase.

Impact

Users might stake tokens in an inactive pool, potentially losing access to rewards or benefits.

Low Severity Issues

Small increase amount does not affect voting power

Resolved

Path

contracts/AlvaStakeProtocol.sol

Function

increaseAmount()

Description

In the increaseAmount function, there is no minimum threshold enforced for the amount being increased. If the user stakes a very small amount, it may not meaningfully impact their voting power due to rounding or small increments being ignored. This can lead to confusion for users and unintended consequences in voting mechanisms where precise voting power adjustments are required.

Recommendation

Add a minimum threshold for the amount parameter in the increaseAmount function to ensure that only significant increases are allowed.

Impact

User Confusion: Users may expect their voting power to increase when they stake, but small increments might have no effect.

Inefficiency: Processing very small amounts can lead to unnecessary gas consumption without tangible benefits.

Improper token burn function used in increaseAmount() and _stake()

Acknowledged

Path

contracts/AlvaStakeProtocol.sol

Description

The increaseAmount() and _stake() functions in the contract use the burnFrom() function of the ALVA contract to burn tokens. However, the burnFrom() function does not have adequate protections in place, such as validating if the caller is authorized or ensuring proper allowances. Instead, the burnTokens() function in the ALVA contract should be used, as it includes necessary protections for secure token burning.

Recommendation

Replace all instances of burnFrom() with burnTokens() in both increaseAmount() and _stake() functions.

Impact

Not using the recommended function (burnTokens()) undermines the standard logic and protections provided by the ALVA contract.

Use CEI pattern

Resolved

Path

contracts/AlvaStakeProtocol.sol

Function

_claimRewards()

Description

In _claimRewards function, the accountToCalculateRewards[] mapping is set to 0 after transfer of tokens which violates the Check Effect Interaction pattern. Which is a necessary part for tackling re-entrancy issues.

Recommendation

Please make sure to set the value to 0 before transferring the tokens.

Informational Severity Issues

Add check for rewardID is present

Resolved

Description

If the user calls claimRewards before the rewards are added then the user won't be earning any rewards. The topUpRewards() function is supposed to be called first before anyone can call claimRewards() function.

Recommendation

To resolve the issue a check can be added where it checks if the reward ID is present and topUpRewards function is called.

Gas Optimisation

Acknowledged

Description

Functions such as veAlvaBalance(), getIncrementedAmount(), getPoolDataByReward() can be set as external to save gas.

Functional Tests

Some of the tests performed are mentioned below:

- testFuzz_StakingFunction()
- test_getRewardsPendingFunction()
- test_stakingFunction()
- test_unstakeFunction()
- test_compoundRewardsFunction()
- test_claimRewardsFunction()
- test_renewStakingFunction
- test_topUpRewardsFunction()
- test_pauseFunction()
- test_unpauseFunction()

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 Alvara staking contract. We performed our audit according to the procedure described above.

Some issues of High,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.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in Alvara staking contract. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of Alvara smart contract. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of the Alvara to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

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.

6+ Years of Expertise	**1M+** Lines of Code Audited
$30B+ Secured in Digital Assets	**1K+** Projects Secured

Follow Our Journey

AUDIT REPORT

December, 2024

For

ALVARA

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com