QuillAudits

Audit Report
October, 2024

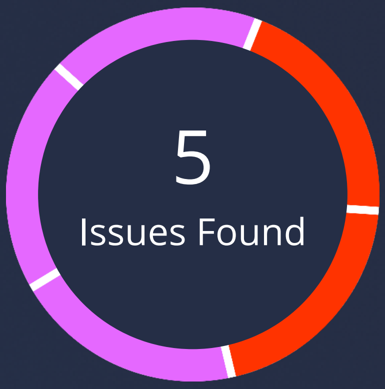
For

Table of Content

Executive Summary

Project Name	Aconomy
Project URL	https://aconomy.io/
Overview	StakingYield.sol contract resembles Synthetix staking rewards contract with yield being paid in the same token as is staked. The Aconomy.sol token contract is a vanilla ERC20 token with the burnable extension added onto it.
Audit Scope	https://github.com/Pandora-Finance/aconomy-contract/tree/AuditFixes
Contracts In-Scope	StakingYield.sol Aconomy.sol
Commit Hash	343e1e18958d2938804e143e3c24ce97fd1a3123
Language	Solidity
Blockchain	Binance Smart Chain
Method	Manual Analysis, Functional Testing, Automated Testing
First Review	10th September 2024 - 12th September 2024
Updated Code Received	1st October 2024
Second Review	15th October 2024 - 16th October 2024
Fixed In	10bd1a81bac97215585c459bce8230f77e2c0d23

Number of Security Issues per Severity



🟥 High	🟨 Medium
🟩 Low	🟪 Informational

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

Checked Vulnerabilities

✓ Access Management	✓ Compiler version not fixed
✓ Arbitrary write to storage	✓ Address hardcoded
✓ Centralization of control	✓ Divide before multiply
✓ Ether theft	✓ Integer overflow/underflow
✓ Improper or missing events	✓ ERC's conformance
✓ Logical issues and flaws	✓ Dangerous strict equalities
✓ Arithmetic Correctness	✓ Tautology or contradiction
✓ Race conditions/front running	✓ Return values of low-level calls
✓ SWC Registry	✓ Missing Zero Address Validation
✓ Re-entrancy	✓ Private modifier
✓ Timestamp Dependence	✓ Revert/require functions
✓ Gas Limit and Loops	✓ Multiple Sends
✓ Exception Disorder	✓ Using suicide
✓ Gasless Send	✓ Using delegatecall
✓ Use of tx.origin	✓ Upgradeable safety
✓ Malicious libraries	✓ Using throw

Checked Vulnerabilities

- ✓ Using inline assembly
- ✓ Style guide violation
- ✓ Unsafe type inference
- ✓ Implicit visibility level

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

Foundry, Slither, Hardhat, Solidity static 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

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

These are the issues identified in the initial audit and have been successfully fixed.

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

1. rewardsToken not updated after an emergency withdraw would impact notifyRewardAmount

Path

StakingYield.sol

Path

recoverERC20(), notifyRewardAmount()

Description

In the case of tokens recovered using recoverERC20(), the ability for the owner to access the deposited tokens at any time would pose a risk that should be promptly addressed. If the owner's account is compromised, the attacker could drain the contract of all funds stored within it.

In the functions, depositRewardTokens() and withdraw(), the variable rewardsToken is used to track the transfers in and out of tokens. In the recoverERC20() function, there is a state change to the number of rewardTokens as well but this is not reflected in the variable. This will cause a break in the token's accounting system as it is used in notifyRewardAmount().

Recommendation

Implement secure practices like using a multi-signature wallet with multiple signers to avoid loss of tokens via onlyOwner controlled functions.

Adjust token accounting in recoverERC20().

Status

Resolved

2. Multiple staking is unhandled by the StakingYield contracts.

Path

StakingYield.sol

Path

stake()

Description

Take a look at stake() function

```
function stake(
    address _userAddress,
    uint256 _amount
) external updateReward(_userAddress) whenNotPaused nonReentrant onlyOwner {
    require(_amount > 0, “amount = 0”);
    Token.transferFrom(msg.sender, address(this), _amount);
    balanceOf[_userAddress] += _amount;
    stakeTimestamps[_userAddress] = block.timestamp;
    totalSupply += _amount;
    emit Staked(_userAddress, _amount);
}
```

Every time a stake is made, it overwrites the stakeTimestamps for the user with the current timestamp. This means that if a user stakes multiple times, only the most recent stake’s timestamp is recorded.
Because only the most recent stake’s timestamp is stored, earlier stakes are treated as if they were made at the same time as the most recent stake. This prevents users from having different lock-up periods for different portions of their staked tokens.
Additionally, while the balanceOf mapping is correctly updated to add new stakes, the contract doesn’t keep track of individual stake amounts and their corresponding timestamps. This makes it impossible to implement a system where different portions of a user’s stake could have different lock-up periods or burning percentages based on when they were staked.

Recommendation

To allow for multiple stakes with different lock-up periods, this would need to redesign the staking mechanism. This could involve creating a struct to store information about each stake and using an array or mapping to keep track of multiple stakes per user.

Client's Comments

Aconomy owner does not expect to stake tokens for one user multiple times.

Status

Acknowledged

Informational Issues

3. Token and burnableToken do not have to be separated

Path

StakingYield.sol

Function

withdraw()

Description

The contract declares two separate state variables for the same token

```
ERC20Burnable public immutable token;

constructor(address _stakingToken) {
    token = ERC20Burnable(_stakingToken);
}
```

Then, use token for all operations, including transfers and burns

```
token.transfer(msg.sender, amount);
token.burn(burnAmount);
```

This change simplifies the contract, reduces gas costs, and eliminates the potential for inconsistencies between two variables representing the same token. The current implementation is unlikely to cause immediate issues. The redundancy mainly affects gas costs and code clarity rather than functionality.

Recommendation

Remove burnableToken variable and refactor parts of the codebase affected.

Status

Resolved

4. Missing zero address check in constructor

Path
StakingYield.sol

Function
constructor()

Description
The constructor of the contract does not check if the provided _stakingToken address is the zero address (0x0). If the zero address is accidentally provided as the _stakingToken:

1. The contract will be deployed with an invalid token address.
2. All subsequent operations involving token transfers or burns will fail.
3. The contract will be rendered unusable and a new deployment will be necessary.

Recommendation
Add a zero address check in the constructor

Status
Resolved

5. General information

Although these findings are not security risks, they represent likely differences in likely user expectations and the provided logic in the contract:

In-contract Timing
Users should be aware that the other contracts in the Aconomy protocol suite (FundingPool, poolAddress, poolRegistry, LibCalculations) operate with a 30-day timeline, counting up to 360 days as a full year. Although this is related to supplying funds or taking loans, and not directly linked to staked tokens, it is worthy to note that 1 year within the StakingYield contract is 365 days.

Partial withdrawals are not allowed
It is not within the scope of the contract to perform partial withdrawal of tokens according to their business logic.

Functional Tests Cases

Some of the tests performed are mentioned below:

- ✓ Stake tokens successfully as the contract owner for a user
- ✓ Attempt to stake 0 tokens (should fail)
- ✓ Attempt to stake as a non-owner address (should fail)
- ✓ Verify correct update of user balance after staking
- ✓ Verify correct update of total supply after staking
- ✓ Withdraw tokens successfully after the minimum staking period
- ✓ Attempt to withdraw before the minimum staking period (should fail)
- ✓ Verify correct burning of tokens based on withdrawal timeframe: (all pass)
  a. Between 1-2 years (50% burn)
  b. Between 2-3 years (25% burn)
  c. After 3 years (no burn)
- ✓ Claim rewards successfully
- ✓ Verify correct update of user's reward balance after claiming
- ✓ Attempt to recover ERC20 tokens as non-owner (should fail)
- ✓ Verify that staking is not possible when paused
- ✓ Stake for multiple users and verify correct reward distribution
- ✓ Verify correct behavior when transitioning between reward periods
- ✓ Owner should recover ERC20 token and update rewardTokens variable/ notifyRewardAmount

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 Aconomy. We performed our audit according to the procedure described above.

Some issues of High, Low and Medium severity were found. Some suggestions, gas optimizations 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 Aconomy. 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 Aconomy. 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 Aconomy 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.

1000+
Audits Completed

$30B
Secured

1M+
Lines of Code Audited

Follow Our Journey

Audit Report
October, 2024

For

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com