ARTOKENS

SMART CONTRACTS REVIEW

zokyo

June 6th 2025 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that these smart contracts passed a security audit.

ZOKYO AUDIT SCORING ARTOKENS GMBH

1. Severity of Issues:
    - Critical: Direct, immediate risks to funds or the integrity of the contract. Typically, these would have a very high weight.
    - High: Important issues that can compromise the contract in certain scenarios.
    - Medium: Issues that might not pose immediate threats but represent significant deviations from best practices.
    - Low: Smaller issues that might not pose security risks but are still noteworthy.
    - Informational: Generally, observations or suggestions that don't point to vulnerabilities but can be improvements or best practices.
2. Test Coverage: The percentage of the codebase that's covered by tests. High test coverage often suggests thorough testing practices and can increase the score.
3. Code Quality: This is more subjective, but contracts that follow best practices, are well-commented, and show good organization might receive higher scores.
4. Documentation: Comprehensive and clear documentation might improve the score, as it shows thoroughness.
5. Consistency: Consistency in coding patterns, naming, etc., can also factor into the score.
6. Response to Identified Issues: Some audits might consider how quickly and effectively the team responds to identified issues.

SCORING CALCULATION:

Let's assume each issue has a weight:
- Critical: -30 points
- High: -20 points
- Medium: -10 points
- Low: -5 points
- Informational: 0 points

Starting with a perfect score of 100:
- 0 Critical issues: 0 points deducted
- 0 High issue: 0 points deducted
- 1 Medium issue: 1 resolved = 0 points deducted
- 2 Low issues: 1 resolved = 0 points deducted
- 1 Informational issue: 1 resolved = 0 points deducted

Thus, the score is 100

TECHNICAL SUMMARY

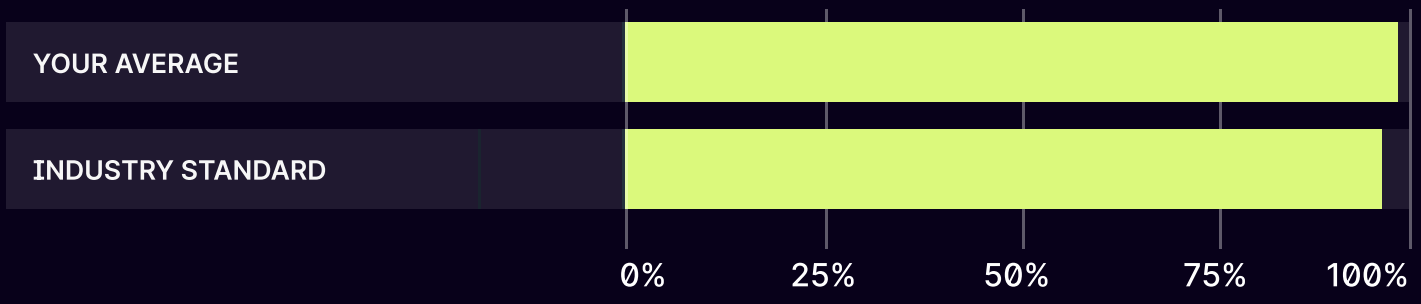
This document outlines the overall security of the ARTokens GmbH smart contract/s evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the ARTokens GmbH smart contract/s codebase for quality, security, and correctness.

Contract Status

There were 0 critical issues found during the review. (See Complete Analysis)

Testable Code



97,44% of the code is testable, which is above the industry standard of 95%.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract/s but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the Ethereum network's fast-paced and rapidly changing environment, we recommend that the ARTokens GmbH team put in place a bug bounty program to encourage further active analysis of the smart contract/s.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the ARTokens repository:
Repo: https://github.com/artokens-fi/erc20-token

Last commit - b7c9877

Contracts under the scope:

- aRToken.sol

During the audit, Zokyo Security ensured that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, inefficient using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most recent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of ARTokens smart contract/s. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were discovered. Part of this work includes writing a test suite using the Foundry testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01	Due diligence in assessing the overall code quality of the codebase.	03	Testing contract/s logic against common and uncommon attack vectors.
02	Cross-comparison with other, similar smart contract/s by industry leaders.	04	Thorough manual review of the codebase line by line.

Executive Summary

ARTokens GmbH is a DeFI protocol which is taking the next great step for our industry as a whole. They intend to make traditional market products available on chain such as ETFs, stocks and commodities with a form of proof of reserves backing every token at 1:1 to be freely traded on the open economy through the various exchanges who accept the ERC20 standard.

Zokyo was tasked with the security review of the ARToken contract responsible for making these financial instruments available on chain. The token adopts the traditional ERC20 standard through the ERC20, ERC20Permit and ERC20Burnable dependencies. In addition to this, AccessControlDefaultAdminRules is relied upon to implement administrative powers.

The default administrator has the authority to set a new proof of reserves data feed, set a new staleness threshold, the minter role has the sole authority to mint tokens and the burner role has the sole authority to remove tokens from the blockchain. This was done to create segregation of powers between each actor responsible for the contract.

Overall the code is well written, well commented and relies on battle hardened libraries which have withstood the test of time. The findings discovered during the audit ranged between Medium and Informational due to the strong adherence to security, the simplicity of the code and reliance of trusted libraries in addition to the avoidance of complicated code. Most of the findings addressed very niche edge case scenarios should certain cases arise and their impacts. Zokyo has created recommendations which will resolve these issues.

It should be noted that the contract administrators must be aware of the implications should a new proof of reserve feed be set within the contract. A new proof of reserve which returns data of different decimal places could have implications on the minting functionality as there is no decimal scaling done within the contract (an issue outlined in this report addresses this edge case scenario). In addition to this, a new proof of reserve with less than expected decimals or tokens available will cause an invalid state where more tokens are available on chain compared to what is available in reserves.

Following the audit, there will be a fix review which will allow the security team to review the fixes made by the developers and retest them to ensure that they address their issues and that they don’t introduce further bugs. Zokyo wishes the team at ARTokens GmbH all the very best when moving to deploy to a production environment.

STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” or “Acknowledged” depending on whether they have been fixed or addressed. Acknowledged means that the issue was sent to the ARTokens GmbH team and the ARTokens GmbH team is aware of it, but they have chosen to not solve it. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

- **Critical**

  The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

- **High**

  The issue affects the ability of the contract to compile or operate in a significant way.

- **Medium**

  The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

- **Low**

  The issue has minimal impact on the contract’s ability to operate.

- **Informational**

  The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

FINDINGS SUMMARY

#	Title	Risk	Status
1	Checks on the new threshold when calling setStalenessThreshold may be too relaxed	Medium	Resolved
2	Insufficient validation on new data feed may cause the contract to accidentally use invalid data	Low	Resolved
3	Using a data feed with different decimals to the old one will break the contract's minting functionality	Low	Resolved
4	AnsweredInRound is deprecated	Informational	Resolved

Checks on the new threshold when calling setStalenessThreshold may be too relaxed

Description:
The `setStalenessThreshold` allows the default admin to update the existing threshold when asserting that the data being returned from the feeds is fresh enough to be used by the minting functionality. This check only ensures that the new threshold is greater than zero but additional checks can also be made to ensure that the new threshold is greater than zero and is within a reasonable range.

Impact:
A threshold too small may cause a denial of service condition within the contract because feeds require a certain amount of time to update but a threshold too great may cause stale data to be returned from the datafeeds. This was rated a medium in severity because whilst this requires an edge case scenario to be met, should the bug trigger, it may cause disruption to the day to day operations of this contract.

Recommendation:
It's recommended that stronger constraints are placed on a new threshold in order to avoid the aforementioned issues. As an example, asserting that the new threshold is greater than zero, is at least 2 hours old but greater than 1 hour will satisfy these constraints.

LOW-1 | RESOLVED

Insufficient validation on new data feed may cause the contract to accidentally use invalid data

Description:
The aRToken contract allows the default admin to set a new feed address via the `setPoRFeedAddress` function in addition to the constructor. This function calls `latestRoundData()` in an attempt to validate the new address; however, additional validation and sanity checks can be performed to ensure the new feed is in fact valid.

Impact:
The contract may call `latestRoundData()` the call may be successful however, invalid data may be returned from the data feed.

Recommendation:
It's recommended that additional sanity checks are made on the data returned by the new data feeds in order to assert that the new feed is in fact valid. For example, checking that answer is greater than zero could be a sufficient way to check that the new feed is alive.

Using a data feed with different decimals to the old one will break the contract's minting functionality

Description:
The default admin has the authority to replace the existing data feed with a new data feed. Chainlink data feeds may return data that is of 8 or 18 decimals which may break the current minting functionality because of the lack of decimal scaling.

Impact:
Should a data feed which differs in decimal places be added to the contract, due to the lack of scaling, this will create an inaccuracy of how many reserve token have been used in comparison to the previous data feed. For example, the token currently uses 18 decimals and a data feed which is of 8 decimals is suddenly added where the previous was also 18 decimals. This may create a denial of service condition because totalAfterMint may well and truly surpass the current reserves when in reality due to the lack of decimal scaling it has not.

Recommendation:
It's recommended that decimal scaling takes place against currentReserves. If the current reserves are of the same decimals as the token, proceed as per normal. If the reserves are of a decimal place less than the token, scale the decimals up to 18 by dividing the result by 10 ** 1e8 and multiplying by 10 ** 1e18.

AnsweredInRound is deprecated

Description:
When getting the current reserves via `getCurrentReserves` the function checks that the `answeredInRound` is greater than or equal to roundId. It should be noted that this return value is deprecated and was previously used when it took multiple rounds to compute the data required.

Src: https://docs.chain.link/data-feeds/api-reference#latestrounddata

	aRToken.sol
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	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

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Security

As a part of our work assisting ARTokens GmbH in verifying the correctness of their contract/s code, our team was responsible for writing integration tests using the Foundry testing framework.
The tests were based on the functionality of the code, as well as a review of the ARTokens GmbH contract/s requirements for details about issuance amounts and how the system handles these.

Ran 14 tests for .audit/test/aRTokenTest.t.sol:aRTokenTest
[PASS] testFuzz_MintAndBurnCycle(uint256,uint256) (runs: 1024, μ: 95378, ~: 95336)
[PASS] testFuzz_MintAndBurnCycle(uint256,uint256,address,address) (runs: 1024, μ: 21026, ~: 19548)
[PASS] testFuzz_MintRejectionWithExcessiveAmounts(uint256) (runs: 1024, μ: 47171, ~: 47488)
[PASS] testFuzz_MintWithValidAmounts(uint256) (runs: 1024, μ: 89949, ~: 90433)
[PASS] testFuzz_MultipleMintsWithinReserves(address[],uint256[]) (runs: 1024, μ: 247067, ~: 189757)
[PASS] testFuzz_ReservesUpdate(int256) (runs: 1024, μ: 106756, ~: 106066)
[PASS] testFuzz_SetStalenessThreshold(uint256) (runs: 1024, μ: 24721, ~: 25380)
[PASS] testFuzz_StalenessThreshold(uint256) (runs: 1024, μ: 26557, ~: 27153)
[PASS] testFuzz_dataStalenessWithTimeJumps(uint256,uint256) (runs: 1024, μ: 119653, ~: 120344)
[PASS] testFuzz_getCurrentReserves(uint80,int256,uint256,uint256,uint80) (runs: 1024, μ: 106974, ~: 97498)
[PASS] testFuzz_setPoRFeedAddress(address) (runs: 1024, μ: 22974, ~: 26636)
[PASS] test_RevertWhen_ConstructorWithZeroPoRFeed() (gas: 120614)
[PASS] test_RevertWhen_ConstructorWithZeroStalenessThreshold() (gas: 122886)
[PASS] test_SetUp() (gas: 42653)

7 passing (376ms)

The resulting code coverage (i.e., the ratio of tests-to-code) is as follows:

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	%UNCOVERED LINES
aRToken.sol	97.44	100	100	97.83	45
All Files	**97.44**	**100**	**100**	**97.83**	

We are grateful for the opportunity to work with the ARTokens GmbH team.

The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.

Zokyo Security recommends the ARTokens GmbH team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo