BLOCKSEC

Security Audit

Report for Aqualoan smart contracts

Date: November 05, 2024 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Aqualoan
Target	Aqualoan smart contracts

Version History

Version	Date	Description
1.0	November 05, 2024	First release

Signature

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 14 million dollars by blocking multiple attacks. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About Target Contracts

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

The target of this audit is the code repository of Aqualoan smart contracts[1] of Aqualoan. Note that, our audit scope is based on the contract addresses on the **BSC Network** provided in the repository[2]. Note the contracts is forked from AAVE V3 core[3] and AAVE V3 peripheral[4]. The forked logic is assume to be reliable and is out of our audit scope.

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version (`Version 1`), as well as new code (in the following versions) to fix issues in the audit report.

Project	Version	Commit Hash
Aqualoan smart contracts	`Version 1`	`324a8197ade6060ea1c8d7948466b6308daab4f3`

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

[1] https://github.com/techaqualoan/aqualoan-smart-contracts/tree/main

[2] https://github.com/techaqualoan/aqualoan-smart-contracts/blob/main/README.md

[3] https://github.com/aave/aave-v3-core

[4] https://github.com/aave/aave-v3-periphery

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Software Security

* Reentrancy
* DoS
* Access control
* Data handling and data flow
* Exception handling
* Untrusted external call and control flow
* Initialization consistency
* Events operation
* Error-prone randomness
* Improper use of the proxy system

1.3.2 DeFi Security

* Semantic consistency
* Functionality consistency
* Permission management
* Business logic
* Token operation
* Emergency mechanism
* Oracle security
* Whitelist and blacklist
* Economic impact
* Batch transfer

1.3.3 NFT Security

* Duplicated item
* Verification of the token receiver
* Off-chain metadata security

1.3.4 Additional Recommendation

* Gas optimization
* Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [5] and Common Weakness Enumeration [6]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Low*
High	High	Medium
Low	Medium	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following four categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Fixed** The item has been confirmed and fixed by the client.

[5] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[6] https://cwe.mitre.org/

Chapter 2 Findings

In total, we found **one** potential security issue and **two** notes.

- Medium Risk: 1
- Note: 2

ID	Severity	Description	Category	Status
1	Medium	Protocol can be drained while creating an empty market	DeFi Security	Confirmed
2	-	Pontential centralization risk	Note	-
3	-	Pontential risk in `stable interestRate` mode	Note	-

The details are provided in the following sections.

2.1 DeFi Security

2.1.1 Protocol can be drained while creating an empty market

Severity Medium

Status Confirmed

Introduced by `Version 1`

Description Due to the use of `rayDiv` in the supply/withdraw process of the Aqualoan protocol, there is no guarantee that the rounding direction always favors the protocol. Attackers can manipulate the `LiquidityIndex` of `Atoken` in an empty market state, exploiting rounding errors to drain assets from the protocol.

```
76   /**
77   * @notice Divides two ray, rounding half up to the nearest ray
78   * @dev assembly optimized for improved gas savings, see https://twitter.com/transmissions11/
         status/1451131036377571328
79   * @param a Ray
80   * @param b Ray
81   * @return c = a raydiv b
82   */
83  function rayDiv(uint256 a, uint256 b) internal pure returns (uint256 c) {
84    // to avoid overflow, a <= (type(uint256).max - halfB) / RAY
85    assembly {
86      if or(iszero(b), iszero(iszero(gt(a, div(sub(not(0), div(b, 2)), RAY))))) {
87        revert(0, 0)
88      }
89
90      c := div(add(mul(a, RAY), div(b, 2)), b)
91    }
92  }
```

Listing 2.1: WadRayMath.sol

Impact The funds in the protocol will be drained.

Suggestion Supply a certain amount of assets in the same transaction of creating a new market.

Feedback from the Project The team will adjust the deployment script to resolve this issue.

2.2 Note

2.2.1 Pontential centralization risk

Introduced by `Version 1`

Description There are some centralization risks in this protocol. For example, the Aqualoan uses upgradeable contracts, which means that if the team's private keys are leaked, malicious users can upgrade the contracts to steal user assets. Additionally, the `PoolAdmin` role can directly transfer assets from the pool and change various parameters in the market, such as LTV, price oracle, and liquidation thresholds, posing risks to user assets. Please ensure that certain key parameters in the on-chain contracts meet the expected values.

Feedback from the Project The team promises to use a multisig wallet.

2.2.2 Pontential risk in `stable interestRate` mode

Introduced by `Version 1`

Description Aave disclosed[1] a vulnerability related to its stable interest rate model and closed the fixed rate mode through a code patch last year. It is recommended to update the corresponding code patch to prevent potential security risks.

Feedback from the Project The team promises to disable the stable interest rate model for all the pools.

[1] https://governance.aave.com/t/aave-v2-v3-security-incident-04-11-2023/15335


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS