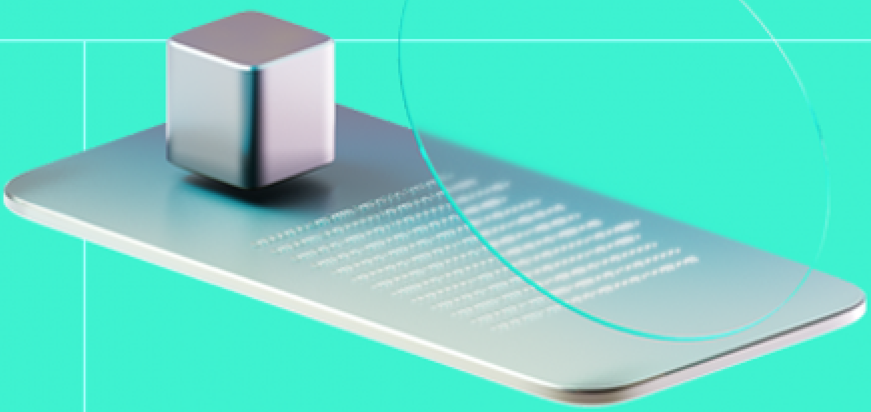
HACKEN

Smart Contract Code Review And Security Analysis Report



Customer: Aqueductfinance

Date: 25/04/2024

We express our gratitude to the Aqueductfinance team for the collaborative engagement that enabled the execution of this Smart Contract Security Assessment.

Aqueduct protocol facilitates controlled token transactions by integrating vesting mechanisms directly into decentralized trading environments. It allows entities to enforce specific vesting schedules upon token sale.

Platform: Etherum

Language: Solidity

Tags: Opensea, Seaport, Vesting, ERC20

Timeline: 23/04/2024 - 24/04/2024

Methodology: https://hackenio.cc/sc_methodology

Review Scope

Repository	https://github.com/aqueduct-finance/otc-v1
Commit	94f8fd8

Audit Summary

9/10	9/10	100%	7/10
Security Score	Code quality score	Test coverage	Documentation quality score

Total 7.7/10

The system users should acknowledge all the risks summed up in the risks section of the report

3	0	0	0
Total Findings	Resolved	Accepted	Mitigated

Findings by severity

Critical	0
High	0
Medium	2
Low	1

Vulnerability	Status
F-2024-1501 - The Contract Should approve(0) First	Pending Fix
F-2024-1504 - Fee-on-Transfer Token Handling Flaw	Pending Fix
F-2024-1533 - Lockup Start Time Accept Past Timestamps	Pending Fix

Document

Name	Smart Contract Code Review and Security Analysis Report for Aqueductfinance
Audited By	Eren Gonen
Approved By	Ataberk Yavuzer
Website	https://aqueduct.finance/
Changelog	25/04/2024 - Preliminary Report

Table of Contents

System Overview

Aqueduct is a Token Sale Protocol that utilizes **OpenSea's** `Seaport` for token settlement and **Hedgey Finance's** `TokenLockupPlansHandler` for vesting with the following contracts:

TokenLockupPlansHandler — A zone contract for **OpenSea's** `Seaport` protocol that atomically creates token lockups post-settlement on `Seaport`, utilizing **Hedgey's** `TokenLockupPlansHandler` contract for the lockups.

Privileged roles

- The Opensea Seaport contract can execute `validateOrder()` after order is settled.

Executive Summary

This report presents an in-depth analysis and scoring of the customer's smart contract project. Detailed scoring criteria can be referenced in the scoring methodology.

Documentation quality

The total Documentation Quality score is **7** out of **10**.

- Functional requirements are provided.
  - NatSpec is sufficient.
- Technical description is partially provided.
  - The interaction of the contract with **Seaport** (e.g., how orders are structured, such as numerator/denominator) is unclear.
  - The interaction of the contract with **TokenLockupPlans** is not clearly documented.

Code quality

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

- The development environment is configured.
- Some best practices are violated.

Test coverage

Code coverage of the project is **100%** (branch coverage).

- The tests are sufficient.

Security score

Upon auditing, the code was found to contain **0** critical, **0** high, **2** medium, and **1** low severity issues, leading to a security score of **8** out of **10**.

All identified issues are detailed in the “Findings” section of this report.

Summary

The comprehensive audit of the customer's smart contract yields an overall score of **7.7**. This score reflects the combined evaluation of documentation, code quality, test coverage, and security aspects of the project.

Risks

- The `TokenLockupPlansHandler` contract interacts externally with the **Opensea** `Seaport v1.5` contract and the **Hedgey Finance** `TokenLockupPlans` contract. Both of these contracts are **out of scope** for this audit. Consequently, any issues that may arise within Seaport or TokenLockupPlans, or from interactions with them, cannot be verified.

Findings

Vulnerability Details

F-2024-1501 - The Contract Should approve(0) First - Medium

Description: Some tokens (like USDT) do not work when changing the allowance from an existing non-zero allowance value. They must first be approved by zero and then the actual allowance must be approved.

Affected lines:

```
116: IERC20(token).approve(address(tokenLockupPlans), amount);
```

Assets:

- contracts/zones/TokenLockupPlansHandler.sol [https://github.com/aqueduct-finance/otc-v1/blob/main/contracts/zones/TokenLockupPlansHandler.sol]

Status: Pending Fix

Classification

Severity: Medium

Impact: 3/5

Likelihood: 3/5

Exploitability: Independent

Complexity: Simple

Likelihood [1-5]: 3

Impact [1-5]: 3

Exploitability [0-2]: 0

Complexity [0-2]: 0

Final Score: 3.0 (Medium)

Hacken Calculator Version: 0.6

Recommendations

Remediation:

When developing functions that update allowances for ERC20 tokens, especially when interacting with tokens known to require this pattern, implement a two-step approval process. This process first sets the allowance for a spender to zero, and then sets it to the desired new value. For example:

```
function resetAndApprove(address token, address spender, uint256 amo
unt) public {
IERC20(token).approve(spender, 0);
IERC20(token).approve(spender, amount);
}
```

F-2024-1504 - Fee-on-Transfer Token Handling Flaw - Medium

Description:

The function below transfers funds from the caller to the receiver via `safeTransferFrom()`, but do not ensure that the actual number of tokens received is the same as the input amount to the transfer. If the token is a fee-on-transfer token, the balance after the transfer will be smaller than expected, leading to accounting issues. Even if there are checks later, related to a secondary transfer, an attacker may be able to use latent funds (e.g. mistakenly sent by another user) in order to get a free credit. One way to address this problem is to measure the balance before and after the transfer, and use the difference as the amount, rather than the stated amount.

Affected code:

```
108: SafeERC20.safeTransferFrom(
109: IERC20(token),
110: recipient,
111: address(this),
112: amount
113: );
```

Assets:

- contracts/zones/TokenLockupPlansHandler.sol [https://github.com/aqueduct-finance/otc-v1/blob/main/contracts/zones/TokenLockupPlansHandler.sol]

Status: Pending Fix

Classification

Severity:	Medium
Impact:	3/5
Likelihood:	3/5
Exploitability:	Independent
Complexity:	Simple

Likelihood [1-5]: 3

Impact [1-5]: 3

Exploitability [0-2]: 0

Complexity [0-2]: 0

Final Score: 3.0 (Medium)

Hacken Calculator Version: 0.6

Recommendations

Remediation:

To mitigate potential vulnerabilities and ensure accurate accounting with fee-on-transfer tokens, modify your contract's token transfer logic to measure the recipient's balance before and after the transfer. Use this observed difference as the actual transferred amount for any further logic or calculations. For example:

```
function transferTokenAndPerformAction(address token, address from,
address to, uint256 amount) public {
uint256 balanceBefore = IERC20(token).balanceOf(to);

// Perform the token transfer
IERC20(token).transferFrom(from, to, amount);

uint256 balanceAfter = IERC20(token).balanceOf(to);
uint256 actualReceived = balanceAfter - balanceBefore;

// Proceed with logic using actualReceived instead of the initial am
ount
require(actualReceived >= minimumRequiredAmount, "Received amount is
less than required");

// Further logic here
}
```

F-2024-1533 - Lockup Start Time Accept Past Timestamps - Low

Description:

The `_createLockup` function within the smart contract facilitates the creation of token lockup plans. This function currently handles `createPlanParams.start`, the parameter indicating the start time for the lockup.

```
function _createLockup(
address recipient,
address token,
uint256 amount,
CreatePlanParams memory createPlanParams
) internal {
...
// if user didn't specify start time, use block.timestamp
if (createPlanParams.start == 0) {
createPlanParams.start = block.timestamp;
}

if (createPlanParams.endOffsetTime < createPlanParams.cliffOffsetTim
e) {
revert END_LESS_THAN_CLIFF();
}

// calculate rate
uint256 rate = (amount * createPlanParams.period) /
createPlanParams.endOffsetTime;

// check for underflow
if (rate == 0) {
revert INVALID_RATE();
}

// create lockup
tokenLockupPlans.createPlan(
recipient,
token,
amount,
createPlanParams.start,
createPlanParams.start + createPlanParams.cliffOffsetTime,
rate,
createPlanParams.period
);
}
```

There exists an oversight where the start time `createPlanParams.start` can be manually set to a timestamp that is in the past (not equal to 0). If `createPlanParams.start` is less than the current `block.timestamp` and is not zero, the lockup plan will erroneously be initialized to a past time. This condition could result in a unintended consequences. Lockup plans set to past timestamps may have their constraints (like vesting periods and cliffs) applied retroactively, potentially resulting in immediate release or availability of tokens that were intended to be locked. The

Assets:

- contracts/zones/TokenLockupPlansHandler.sol [https://github.com/aqueduct-finance/otc-v1/blob/main/contracts/zones/TokenLockupPlansHandler.sol]

Status:	Pending Fix

Classification

Severity:	Low
Impact:	2/5
Likelihood:	3/5
Exploitability:	Independent
Complexity:	Simple

Likelihood [1-5]: 3

Impact [1-5]: 2

Exploitability [0-2]: 0

Complexity [0-2]: 0

Final Score: 2.5 (Low)

Recommendations

Remediation: Introduce a validation step in the `_createLockup` function to explicitly check if `createPlanParams.start` is set to a past timestamp either use `block.timestamp` or revert.

Observation Details

F-2024-1503 - Solidity version 0.8.20 might not work on all chains due to `PUSH0` - Info

Description: The Solidity version 0.8.20 employs the recently introduced `PUSH0` opcode in the Shanghai EVM. This opcode might not be universally supported across all blockchain networks and Layer 2 solutions. Thus, as a result, it might be not possible to deploy solution with version 0.8.20 >= on some blockchains.

```
pragma solidity 0.8.20;
```

Assets:

- contracts/zones/TokenLockupPlansHandler.sol [https://github.com/aqueduct-finance/otc-v1/blob/main/contracts/zones/TokenLockupPlansHandler.sol]

Status: Pending Fix

Recommendations

Remediation: It is recommended to verify whether solution can be deployed on particular blockchain with the Solidity version 0.8.20 >=. Whenever such deployment is not possible due to lack of `PUSH0` opcode support and lowering the Solidity version is a must, it is strongly advised to review all feature changes and bugfixes in [Solidity releases] (https://soliditylang.org/blog/category/releases/). Some changes may have impact on current implementation and may impose a necessity of maintaining another version of solution.

F-2024-1505 - Floating pragma - Info

Description: The project uses floating pragma `^0.8.20`.

This may result in the contracts being deployed using the wrong pragma version, which is different from the one they were tested with. For example, they might be deployed using an outdated pragma version which may include bugs that affect the system negatively.

Assets:

- contracts/zones/TokenLockupPlansHandler.sol [https://github.com/aqueduct-finance/otc-v1/blob/main/contracts/zones/TokenLockupPlansHandler.sol]

Status: Pending Fix

Recommendations

Remediation: Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment. Consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.

F-2024-1506 - Missing Zero Address Validation - Info

Description:

In Solidity, the Ethereum address `0x0000000000000000000000000000000000000000` is known as the "**zero address**". This address has significance because it is the default value for uninitialized address variables and is often used to represent an invalid or non-existent address.

The "**Missing zero address control**" issue arises when a Solidity smart contract does not properly check or prevent interactions with the zero address, leading to unintended behavior.

For instance, a contract might allow tokens to be sent to the zero address without any checks, which essentially burns those tokens as they become irretrievable. While sometimes this is intentional, without proper control or checks, accidental transfers could occur.

Missing check were observed in the following contract:

- `./TokenLockupPlansHandler.sol: constructor()`

Assets:

- contracts/zones/TokenLockupPlansHandler.sol [https://github.com/aqueduct-finance/otc-v1/blob/main/contracts/zones/TokenLockupPlansHandler.sol]

Status: Pending Fix

Recommendations

Remediation: Implement zero address validation for the given parameters. This can be achieved by adding require statements that ensure address parameters are not the zero address.

F-2024-1507 - Unchecked Return Value from approve() Function in createLockup() - Info

Description: The `_createLockup()` function currently lacks a step in its implementation by not verifying the return value of the call to the `approve()` function. This omission introduces a potential risk, as the success or failure of the approval transaction is not being validated, which can lead to unforeseen issues and jeopardize the intended functionality of the contract.

```
// approve tokenLockupPlans contract to spend this token
IERC20(token).approve(address(tokenLockupPlans), amount);
// if user didn't specify start time, use block.timestamp
if (createPlanParams.start == 0) {
createPlanParams.start = block.timestamp;
}
if (createPlanParams.endOffsetTime < createPlanParams.cliffOffsetTim
e) {
revert END_LESS_THAN_CLIFF();
}
```

Assets:

- contracts/zones/TokenLockupPlansHandler.sol [https://github.com/aqueduct-finance/otc-v1/blob/main/contracts/zones/TokenLockupPlansHandler.sol]

Status: Pending Fix

Recommendations

Remediation: Use OpenZeppelin's `increaseAllowance()` function from safeERC20 library or validate the approve function's return value.

F-2024-1508 - Redundant import - Info

Description:

The **ConsiderationStructs.sol** contract is imported redundantly in two separate lines, which could be consolidated into a single import statement. This change would simplify the code and improve its readability.

Affected Code:

```
import {ZoneParameters, Schema} from "seaport-types/src/lib/ConsiderationStructs.sol";
import {ITokenLockupPlans} from "../misc/interfaces/ITokenLockupPlans.sol";
import {SpentItem, ReceivedItem} from "seaport-types/src/lib/ConsiderationStructs.sol";
```

Assets:

- contracts/zones/TokenLockupPlansHandler.sol [https://github.com/aqueduct-finance/otc-v1/blob/main/contracts/zones/TokenLockupPlansHandler.sol]

Status: Pending Fix

Recommendations

Remediation: Consolidate the import statements into a single line.

F-2024-1513 - Use '<=' Operator in Time Validation Checks for Enhanced Flexibility - Info

Description: In the smart contract function `_createLockup`, comparison between `endOffsetTime` and `cliffOffsetTime`. The function utilizes the < operator to ensure that `endOffsetTime` is not less than `cliffOffsetTime`:

```
if (createPlanParams.endOffsetTime < createPlanParams.cliffOffsetTim
e) {
revert END_LESS_THAN_CLIFF();
}
```

This condition is intended to prevent the creation of a lockup plan where the end time precedes the cliff time, the use of the < operator restricts the creation of plans where the `endOffsetTime` is exactly equal to the `cliffOffsetTime`. Allowing `endOffsetTime` to be equal to `cliffOffsetTime` could offer more flexibility in setting up lockup plans, especially in scenarios where a single-point time lockup might be desired.

Assets:

- contracts/zones/TokenLockupPlansHandler.sol [https://github.com/aqueduct-finance/otc-v1/blob/main/contracts/zones/TokenLockupPlansHandler.sol]

Status: Pending Fix

Recommendations

Remediation: Replace the < operator with the <= operator in the time validation check.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing 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 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, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Appendix 1. Severity Definitions

When auditing smart contracts, Hacken is using a risk-based approach that considers **Likelihood**, **Impact**, **Exploitability** and **Complexity** metrics to evaluate findings and score severities.

Reference on how risk scoring is done is available through the repository in our Github organization:

hknio/severity-formula

Severity	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.
Medium	Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.
Low	Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution, do not affect security score but can affect code quality score.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Scope Details

Repository	https://github.com/aqueduct-finance/otc-v1/tree/main
Commit	94f8fd84bae9e7c073cfe31c9385c0c526077e6e
Whitepaper	N/A
Requirements	N/A
Technical Requirements	https://github.com/aqueduct-finance/otc-v1/blob/main/README.md

Contracts in Scope

./contracts/TokenLockupPlansHandler.sol