F Y E O

Security Assessment of the Deepwaters Solidity Contracts

VATNFORN Corp.

January 2023
Version 1.0

Presented by:

FYEO Inc.

PO Box 147044
Lakewood CO 80214
United States

Security Level
Strictly Confidential

Table of Contents

List of Figures

List of Tables

EXECUTIVE SUMMARY

Overview

VATNFORN Corp. engaged FYEO Inc. to perform a Security Assessment of the Deepwaters Solidity Contracts.

The assessment was conducted remotely by the FYEO Security Team. Testing took place on January 03 - January 09, 2023, and focused on the following objectives:

- To provide the customer with an assessment of their overall security posture and any risks that were discovered within the environment during the engagement.
- To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place.
- To identify potential issues and include improvement recommendations based on the results of our tests.

This report summarizes the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the FYEO Security Team took to identify and validate each issue, as well as any applicable recommendations for remediation.

Key Findings

The following issues were identified during the testing period. These were since remediated:

- FYEO-DP-01 – Different locked pragma versions used Vault and Swap
- FYEO-DP-02 – Known issues in the compiler

Based on our review process, we conclude that the reviewed code implements the documented functionality.

Scope and Rules of Engagement

The FYEO Review Team performed a Security Assessment of the Deepwaters Solidity Contracts. The following table documents the targets in scope for the engagement. No additional systems or resources were in scope for this assessment.

The source code was supplied through a private repository at https://github.com/deepwatersxyz/deepwaters-tokens with the commit hash adaad1e07ea1477e4ee1c76510a32216254bf8ec.

Re-review was carried out with the commit hash b10910c9a7b5d3986c7307df42734f1885d527cb

Files included in the code review

deepwaters/
└── contracts/
    ├── swap/
    │   ├── positionManager.sol
    │   ├── rebalancerUpgradable.sol
    ├── tokens/
    │   │   ├── WTR/
    │   │   │   ├── WTRToken.sol

Table 1: Scope

TECHNICAL ANALYSES AND FINDINGS

During the Security Assessment of the Deepwaters Solidity Contracts, we discovered:

- 1 finding with MEDIUM severity rating.
- 1 finding with INFORMATIONAL severity rating.

The following chart displays the findings by severity.

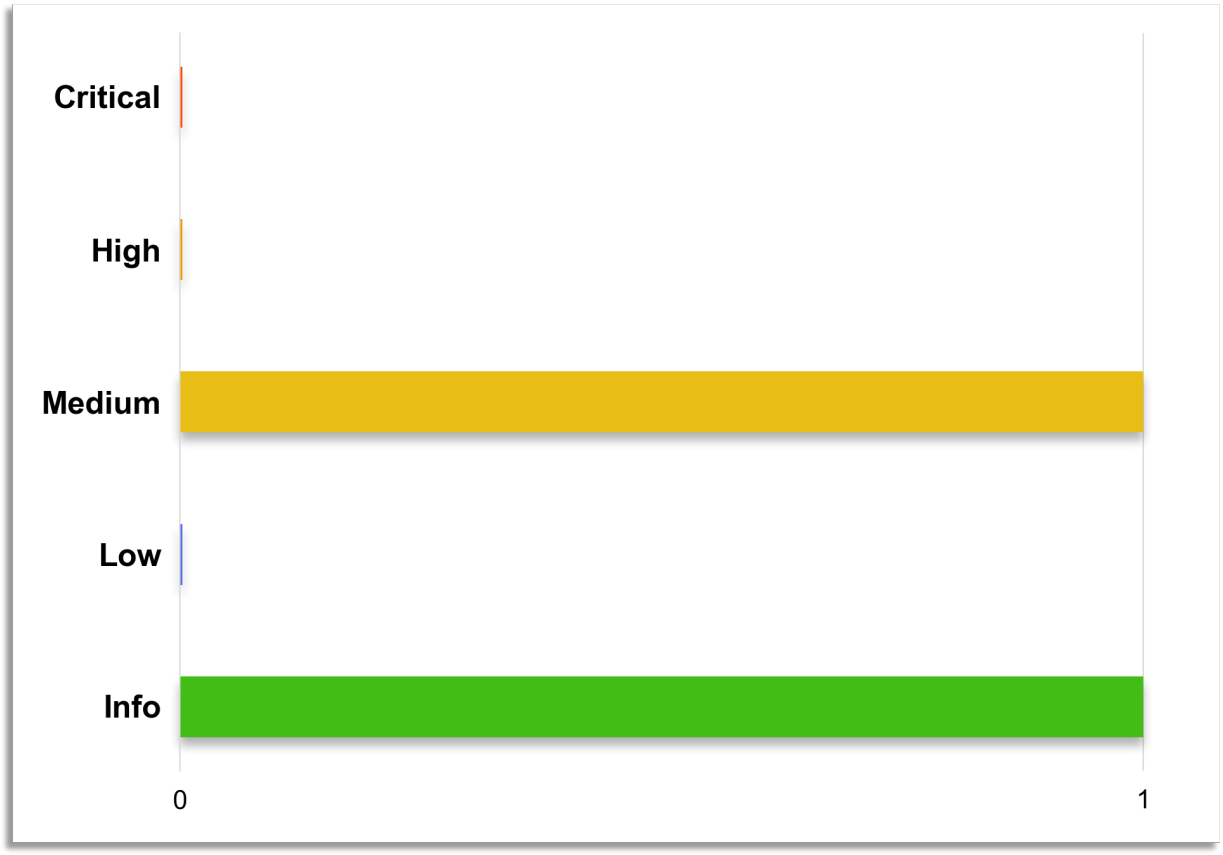

Figure 1: Findings by Severity

FINDINGS

The *Findings* section provides detailed information on each of the findings, including methods of discovery, explanation of severity determination, recommendations, and applicable references.

The following table provides an overview of the findings.

Finding #	Severity	Description
FYEO-DP-01	Medium	Different locked pragma versions used Vault and Swap
FYEO-DP-02	Informational	Known issues in compiler

Table 2: Findings Overview

TECHNICAL ANALYSIS

The source code has been manually validated to the extent that the state of the repository allowed. The validation includes confirming that the code correctly implements the intended functionality.

Technical Findings

General Observations

Deepwater has implemented a trustless exchange in such a way that its attack surface is limited relating to the swap and token functionality of the contracts in scope. The code is well-structured and follows security best practices and uses up-to-date and hardened libraries.

Exchanges typically suffer from problems relating to order flow integrity and the comingling of customer funds that leave many users open to black swan events or exchange operator malfeasance. The architecture allows for a solution to the aforementioned issues which allows multiple EVM native tokens to be deposited and subsequently traded.

Deepwater's implementation utilizes varying fixed versions of the solidity compiler. The compiler version and flag differences between the token and swap contracts can pose issues. There are also known issues with the compiler being used, more details are available in the body of the report.

FYEO's perspective is that there are no inherent risks from an infrastructure point of view as long as sufficient safeguards are put in place.

DIFFERENT LOCKED PRAGMA VERSIONS USED VAULT AND SWAP

Finding ID: FYEO-DP-01
Severity: Medium
Status: Remediated

Description

In the orginal commit hash there were different compiler versions used between the vault and swap functionality. Versions were locked following best practices but were using different minor versions.

Proof of Issue

File name: contracts/swap/PositionManager.sol
contracts/tokens/WTR

Line number: 2

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.4;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
import "@openzeppelin/contracts/utils/cryptography/draft-EIP712.sol";
```

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.10;

import "@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol";
import "@openzeppelin/contracts-
upgradeable/access/AccessControlEnumerableUpgradeable.sol";
import "@openzeppelin/contracts-
upgradeable/token/ERC20/extensions/ERC20BurnableUpgradeable.sol"
```

Severity and Impact Summary

Different compiler versions can negatively affect the contract system and have the potential to introduce bugs into the code.

Recommendation

It is recommended to lock fixed pragma versions to a specific version

KNOWN ISSUES IN COMPILER

Finding ID: FYEO-DP-02
Severity: **Informational**
Status: **Remediated**

Description

There are several issues that are known to be present in the solc 0.8.10, with some being introduced in this version.

Proof of Issue

File name: hardhat.config.ts

Line number: 111

```
          version: '0.8.10',
        settings: {
          optimizer: {
            enabled: true,
            runs: 200,
          },
          metadata: {
            bytecodeHash: 'none',
```

Severity and Impact Summary

Solc `0.8.10` is affected by the following issues

- SOL-2022-2: NestedCallataArrayAbiReencodingSizeValidation (very low)
- SOL-2022-3: DataLocationChangeInInternalOverride (very low)
- SOL-2022-5: DirtyBytesArrayToStorage (low)
- SOL-2022-6: AbiReencodingHeadOverflowWithStaticArrayCleanup (medium)

Recommendation

It is recommended to consider upgrading to version 0.8.14.

References

- https://docs.soliditylang.org/en/latest/bugs.html

OUR PROCESS

METHODOLOGY

FYEO Inc. uses the following high-level methodology when approaching engagements. They are broken up into the following phases.

Figure 2: Methodology Flow

KICKOFF

The project is kicked off as the sales process has concluded. We typically set up a kickoff meeting where project stakeholders are gathered to discuss the project as well as the responsibilities of participants. During this meeting we verify the scope of the engagement and discuss the project activities. It's an opportunity for both sides to ask questions and get to know each other. By the end of the kickoff there is an understanding of the following:

- Designated points of contact
- Communication methods and frequency
- Shared documentation
- Code and/or any other artifacts necessary for project success
- Follow-up meeting schedule, such as a technical walkthrough
- Understanding of timeline and duration

RAMP-UP

Ramp-up consists of the activities necessary to gain proficiency on the project. This can include the steps needed for familiarity with the codebase or technological innovation utilized. This may include, but is not limited to:

- Reviewing previous work in the area including academic papers
- Reviewing programming language constructs for specific languages
- Researching common flaws and recent technological advancements

Review

The review phase is where most of the work on the engagement is completed. This is the phase where we analyze the project for flaws and issues that impact the security posture. Depending on the project this may include an analysis of the architecture, a review of the code, and a specification matching to match the architecture to the implemented code.

In this code audit, we performed the following tasks:

1. Security analysis and architecture review of the original protocol
2. Review of the code written for the project
3. Compliance of the code with the provided technical documentation

The review for this project was performed using manual methods and utilizing the experience of the reviewer. No dynamic testing was performed, only the use of custom-built scripts and tools were used to assist the reviewer during the testing. We discuss our methodology in more detail in the following sections.

Code Safety

We analyzed the provided code, checking for issues related to the following categories:

- General code safety and susceptibility to known issues
- Poor coding practices and unsafe behavior
- Leakage of secrets or other sensitive data through memory mismanagement
- Susceptibility to misuse and system errors
- Error management and logging

This list is general and not comprehensive, meant only to give an understanding of the issues we are looking for.

Technical Specification Matching

We analyzed the provided documentation and checked that the code matches the specification. We checked for things such as:

- Proper implementation of the documented protocol phases
- Proper error handling
- Adherence to the protocol logical description

Reporting

FYEO Inc. delivers a draft report that contains an executive summary, technical details, and observations about the project.

The executive summary contains an overview of the engagement including the number of findings as well as a statement about our general risk assessment of the project. We may conclude that the overall risk is low but depending on what was assessed we may conclude that more scrutiny of the project is needed.

We report security issues identified, as well as informational findings for improvement, categorized by the following labels:

- Critical
- High
- Medium
- Low
- Informational

The technical details are aimed more at developers, describing the issues, the severity ranking and recommendations for mitigation.

As we perform the audit, we may identify issues that aren't security related, but are general best practices and steps that can be taken to lower the attack surface of the project. We will call those out as we encounter them and as time permits.

As an optional step, we can agree on the creation of a public report that can be shared and distributed with a larger audience.

Verify

After the preliminary findings have been delivered, this could be in the form of the approved communication channel or delivery of the draft report, we will verify any fixes within a window of time specified in the project. After the fixes have been verified, we will change the status of the finding in the report from open to remediated.

The output of this phase will be a final report with any mitigated findings noted.

Additional Note

It is important to note that, although we did our best in our analysis, no code audit or assessment is a guarantee of the absence of flaws. Our effort was constrained by resource and time limits along with the scope of the agreement.

While assessing the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. This is a solid baseline for severity determination.

THE CLASSIFICATION OF VULNERABILITIES

Security vulnerabilities and areas for improvement are weighted into one of several categories using, but is not limited to, the criteria listed below:

Critical – vulnerability will lead to a loss of protected assets

- This is a vulnerability that would lead to immediate loss of protected assets
- The complexity to exploit is low
- The probability of exploit is high

High - vulnerability has potential to lead to a loss of protected assets

- All discrepancies found where there is a security claim made in the documentation that cannot be found in the code
- All mismatches from the stated and actual functionality
- Unprotected key material
- Weak encryption of keys
- Badly generated key materials
- Txn signatures not verified
- Spending of funds through logic errors
- Calculation errors overflows and underflows

Medium - vulnerability hampers the uptime of the system or can lead to other problems

- Insecure calls to third party libraries
- Use of untested or nonstandard or non-peer-reviewed crypto functions
- Program crashes, leaves core dumps or writes sensitive data to log files

Low – vulnerability has a security impact but does not directly affect the protected assets

- Overly complex functions
- Unchecked return values from 3rd party libraries that could alter the execution flow

Informational

- General recommendations