BLOCKAPEX

SMART CONTRACT SECURITY

V 1.0

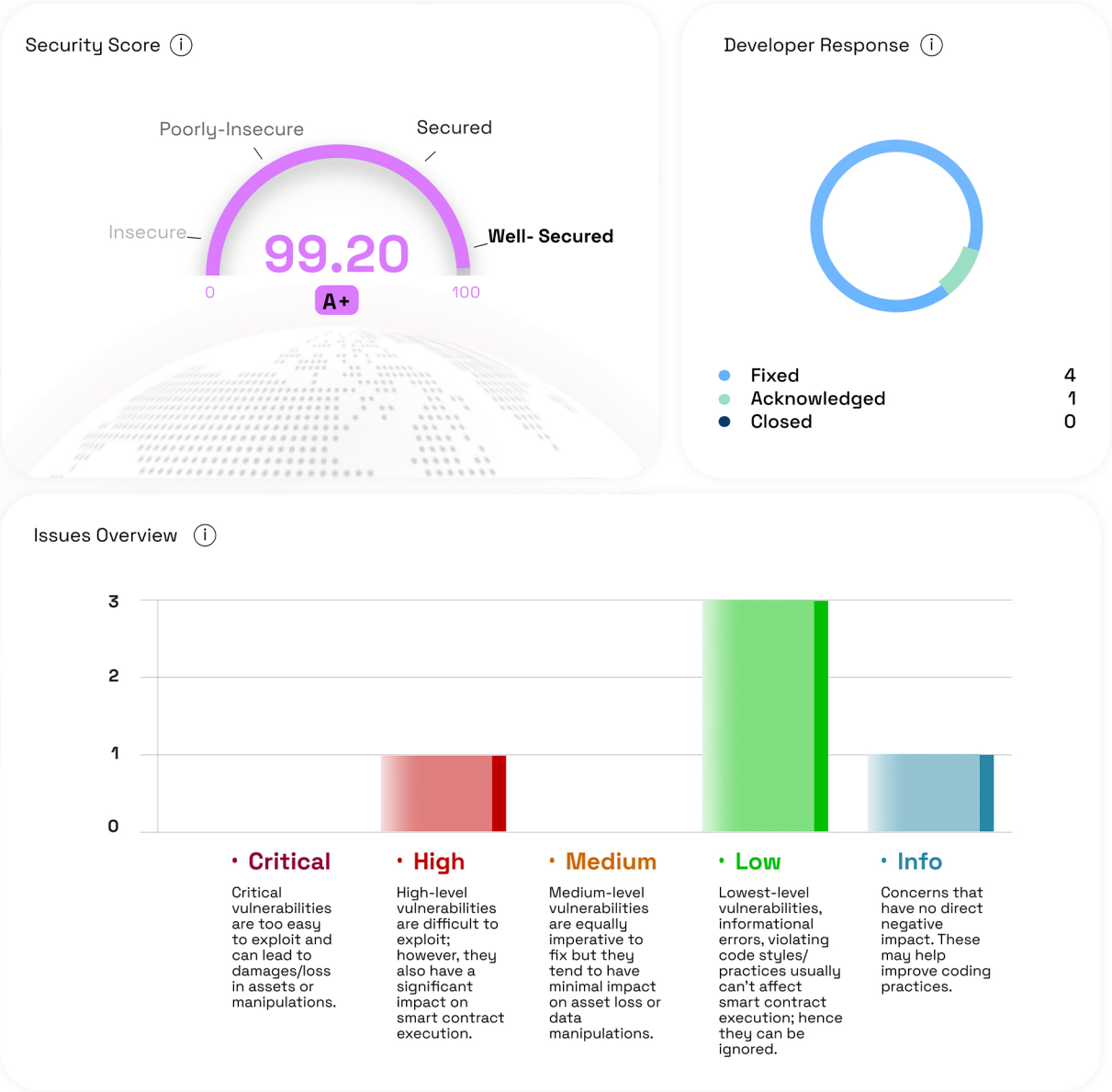
DATE: 23rd JAN 2024

PREPARED FOR: ZERO SWAP

Contents

1 Executive Summary

Zero Liquid engaged BlockApex for a security review of the Zero Liquid Protocol's feature "Swap". A team of two security researchers reviewed the source code of the Swap feature for 4-days of effort. The details of project scope, complexity and coverage are provided in the the following sections of this report.



1.1 Scope

1.1.1 In Scope

Overview of the ZeroLiquidSwap contract: Swap is a Zero Liquid protocol's feature implemented as a single smart contract written in Solidity. The contract itself comprises of two externally visible methods and two interfaces facilitating interactions with the Curve's Stable Swap and the 1Inch's Swap Aggregator. Key functionalities include deposits and swaps, allowing users to register and exchange the zETH token routing via renowned aggregators and DExes.

Files in scope: src/ZeroLiquidSwap.sol

Initial Audit Commit Hash: 1a2c98b6c68c7f95480eb8c241a265318c401136

Final Audit Commit Hash: 2459f79c65c793d00cd2a4b904354b72ead22e66

Note: Any external calls to composable smart contracts are assumed to be safe.

1.1.2 Out of Scope

Any features or functionalities not explicitly mentioned in the "In Scope" section are considered outside the scope of this security review.

1.2 Methodology

The codebase was audited using a filtered audit technique. A band of two (2) auditors scanned the codebase in an iterative process for a time spanning 4 days. Starting with the recon phase, a basic understanding of the code was developed, and the auditors worked on developing presumptions for the developed codebase and the relevant documentation/ whitepaper. Furthermore, the audit moved on with the manual code reviews to find logical flaws in the codebase complemented with code optimizations, software, and security design patterns, code styles and best practices.

Status Description

Acknowledged: The issue has been recognized and is under review. It indicates that the relevant team is aware of the problem and is actively considering the next steps or solutions.

Fixed: The issue has been addressed and resolved. Necessary actions or corrections have been implemented to eliminate the vulnerability or problem.

Closed: This status signifies that the issue has been thoroughly evaluated and acknowledged by the development team. While no immediate action is being taken.

2 Summary of Findings

S.No	Severity	Findings	Status
#1	HIGH	Desynchronized State Leads to Locked Funds	FIXED
#2	LOW	Inexistent Sanitization of Input Addresses	FIXED
#3	LOW	Inexistent Event Emission	FIXED
#4	LOW	Inexistent Setter for Updates	ACKNOWLEDGED
#5	INFO	Insufficient NatSpec Documentation	FIXED

3 Findings and Risk Analysis

3.1 Desynchronized State Leads to Locked Funds

Severity: High

Status: Fixed

Location :

1. src/ZeroLiquidSwap.sol#L121-L139

Description In the `swap` function of the `ZeroLiquidSwap` contract, the contract neglects to check the return values from the external contract call to `IStableSwap.exchange`. This shortcoming may be proved critical where the underlying assumption to the Curve's Stable Swap pool that enables the conversion of `zETH` to `wETH` is broken.

The issue arises when, a user tries to swap their deposited `yield tokens` to the `desired tokens`, the function does not incorporate the differences between expected and actual values exchanged on the Curve's Stable Swap which will result in locked funds for the user. For the cases where `zeth/weth` exchange may require less `zETH` deposited to give more `wETH`, the system locks the additional `wETH` swapped from Curve's Stable Swap pool.

Impact The absence of checks on these return values may result in a state where the contract believes an action has succeeded when it has not, leading to discrepancies in contract state and potentially causing financial losses or inconsistencies in the accounting of assets. Additionally, in scenarios where subsequent contract logic depends on the successful completion of these calls, unchecked failures could propagate errors or vulnerabilities further into the contract's execution flow.

Proof of Concept

```
        function swap(...) external {
            IZeroLiquid(zeroliquid).mintFrom(msg.sender, debtAmount, address(this));

            SafeERC20.safeApprove(debtToken, stableSwap, debtAmount);
            IStableSwap(stableSwap).exchange(zethPoolIndex, wethPoolIndex, debtAmount,
                minDebtExchangeAmount);

            // Give approval to 1inch&#x27;s AggregationRouterV5
            SafeERC20.safeApprove(desc.srcToken, swapRouter, desc.amount);
            IAggregationRouterV5(swapRouter).swap{ value: 0 }(executor, desc, permit, data);
        }
```

Recommendation It is recommended to modify the `SwapDescription` struct on the go, where the exchanged value of wETH for zETH is stored in a variable and set as the `desc.amount` to be swapped for the accurate values instead of expected vs actual values as currently implemented.

3.2 Inexistent Sanitization of Input Addresses

Severity: Low

Status: Fixed

Location :

1. src/ZeroLiquidSwap.sol#L53-L67

Description The `ZeroLiquidSwap` smart contract does not implement sanity checks on the constructor where incoming function arguments are not checked for `zero values`. This practice is suboptimal as faulty deployments can incur unnecessary gas cost for the Zero Liquid Protocol.

Proof of Concept

```
        constructor(
            address _zeroliquid,
            address _debtToken,
            address _swapRouter,
            address _stableSwap,
            int128 _wethPoolIndex,
            int128 _zethPoolIndex
        ) {
            zeroliquid = _zeroliquid;
            debtToken = _debtToken;
            swapRouter = _swapRouter;
            stableSwap = _stableSwap;
            wethPoolIndex = _wethPoolIndex;
            zethPoolIndex = _zethPoolIndex;
        }
```

Recommendation :

- Short term: Ensure the deployments are done through scripts with verified input arguments.
- Long term: Some basic sanitization to be put in place by ensuring that each address specified is non-zero.

3.3 Inexistent Event Emission

Severity: Low

Status: Fixed

Location :

1. deposit() - src/ZeroLiquidSwap.sol#L106
2. swap() - src/ZeroLiquidSwap.sol#L139

Description The contract lacks event emissions in its `deposit` and `swap` functions. Emitting events is a best practice in smart contract development, as it helps in tracking and verifying contract interactions on the blockchain.

Proof of Concept

```
        function swap(
            uint256 debtAmount,
            uint256 minDebtExchangeAmount,
            address executor,
            IAggregationRouterV5.SwapDescription calldata desc,
            bytes calldata permit,
            bytes calldata data
        )
            external
        {...}

        function deposit(
            address recipient,
            address executor,
            IAggregationRouterV5.SwapDescription calldata desc,
            bytes calldata permit,
            bytes calldata data
        )
            external
            payable
        {...}
```

Recommendation :

1. Implement event emissions in both functions to log significant actions, such as successful deposits and swaps. This will enhance traceability and transparency of the contract's operations.

3.4 Inexistent Setter for Updates

Severity: Low

Status: Acknowledged

Location :

- zeroLiquid: src/ZeroLiquidSwap.sol#L44
- swapRouter: src/ZeroLiquidSwap.sol#L49

Description The contract's addresses for `swapRouter` and `zeroLiquid` are immutable and lack setter functions. This design choice restricts the contract's ability to adapt to changes, like updating to a new router or the zero liquid protocol itself, in case of vulnerabilities or upgrades in dependent contracts.

Impact The rigidity of immutable addresses limits the contract's flexibility and may necessitate redeployment if any of the referenced contracts require updates.

Proof of Concept

```
        address public immutable zeroliquid;


        // 1inch AggregationRouterV5 address
        address public immutable swapRouter;
```

Recommendation :

1. Short term: Evaluate the necessity of immutability for these addresses and indices.
2. Long term: Consider implementing controlled, role-based setter functions to allow updates to these addresses in a secure manner. This approach can balance the need for contract stability with adaptability to changing circumstances or improvements in the broader ecosystem.

3.5 Insufficient NatSpec Documentation

Severity: Info

Status: Fixed

Description Throughout the smart contract, the interfaces `IZeroLiquid` and `IAggregationRouterV5` are devoid of NatSpec comments. This oversight in documentation creates a barrier to transparency, hiding the nuances and operational intricacies of functions and hindering the ability of developers to grasp the contract's design intent and the potential interactions between its functions and external contracts.

Recommendation Implement detailed Natspec comments for all interfaces and their functions. This practice will significantly enhance the contract's clarity, auditability, and user confidence.

Disclaimer:

The smart contracts provided by the client with the purpose of security review have been thoroughly analyzed in compliance with the industrial best practices till date w.r.t. Smart Contract Weakness Classification (SWC) and Cybersecurity Vulnerabilities in smart contract code, the details of which are enclosed in this report.

This report is not an endorsement or indictment of the project or team, and they do not in any way guarantee the security of the particular object in context. This report is not considered, and should not be interpreted as an influence, on the potential economics of the token (if any), its sale, or any other aspect of the project that contributes to the protocol's public marketing.

Crypto assets/ tokens are the results of the emerging blockchain technology in the domain of decentralized finance and they carry with them high levels of technical risk and uncertainty. No report provides any warranty or representation to any third-party in any respect, including regarding the bug-free nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the reports in any way, including to make any decisions to buy or sell any token, product, service, or asset. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and is not a guarantee as to the absolute security of the project.

Smart contracts are deployed and executed on a blockchain. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. The scope of our review is limited to a review of the programmable code and only the programmable code, we note, as being within the scope of our review within this report. The smart contract programming language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer or any other areas beyond the programming language's compiler scope that could present security risks.

This security review cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While BlockApex has done their best in conducting the analysis and producing this report, it is important to note that one should not rely on this report only - we recommend proceeding with several independent code security reviews and a public bug bounty program to ensure the security of smart contracts.