APR.24

SECURITY REVIEW REPORT FOR 1INCH

ABOUT HEXENS

Hexens is a cybersecurity company that strives to elevate the standards of security in Web 3.0, create a safer environment for users, and ensure mass Web 3.0 adoption.

Hexens has multiple top-notch auditing teams specialized in different fields of information security, showing extreme performance in the most challenging and technically complex tasks, including but not limited to: Infrastructure Audits, Zero Knowledge Proofs / Novel Cryptography, DeFi and NFTs. Hexens not only uses widely known methodologies and flows, but focuses on discovering and introducing new ones on a day-to-day basis.

In 2022, our team announced the closure of a $4.2 million seed round led by IOSG Ventures, the leading Web 3.0 venture capital. Other investors include Delta Blockchain Fund, Chapter One, Hash Capital, ImToken Ventures, Tenzor Capital, and angels from Polygon and other blockchain projects.

Since Hexens was founded in 2021, it has had an impressive track record and recognition in the industry: Mudit Gupta - CISO of Polygon Technology - the biggest EVM Ecosystem, joined the company advisory board after completing just a single cooperation iteration. Polygon Technology, 1inch, Lido, Hats Finance, Quickswap, Layerswap, 4K, RociFi, as well as dozens of DeFi protocols and bridges, have already become our customers and taken proactive measures towards protecting their assets.

CONTENTS

SCOPE

The analyzed resources are located on:

https://github.com/1inch/limit-order-protocol/pull/306/files

https://github.com/1inch/limit-order-settlement/compare/2.0.0-prerelease-2...master

AUDITING DETAILS

STARTED

01.04.2024

DELIVERED

08.04.2024

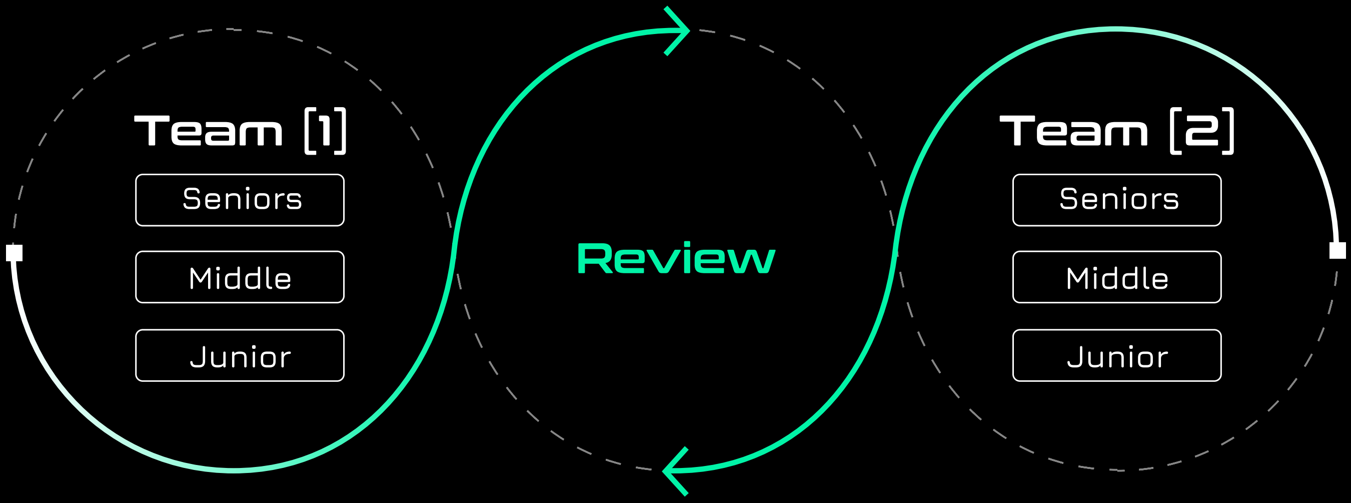
Review Led by

KASPER ZWIJSEN

Head of Smart Contract Audits | Hexens

HEXENS METHODOLOGY

Hexens methodology involves 2 teams, including multiple auditors of different seniority, with at least 5 security engineers. This unique cross-checking mechanism helps us provide the best quality in the market.



SEVERITY STRUCTURE

The vulnerability severity is calculated based on two components

- Impact of the vulnerability
- Probability of the vulnerability

Impact	Probability			
	rare	unlikely	likely	very likely
Low/Info	Low/Info	Low/Info	Medium	Medium
Medium	Low/Info	Medium	Medium	High
High	Medium	Medium	High	Critical
Critical	Medium	High	Critical	Critical

SEVERITY CHARACTERISTICS

Smart contract vulnerabilities can range in severity and impact, and it's important to understand their level of severity in order to prioritize their resolution. Here are the different types of severity levels of smart contract vulnerabilities:

Critical

Vulnerabilities with this level of severity can result in significant financial losses or reputational damage. They often allow an attacker to gain complete control of a contract, directly steal or freeze funds from the contract or users, or permanently block the functionality of a protocol. Examples include infinite mints and governance manipulation.

High

Vulnerabilities with this level of severity can result in some financial losses or reputational damage. They often allow an attacker to directly steal yield from the contract or users, or temporarily freeze funds. Examples include inadequate access control integer overflow/underflow, or logic bugs.

Medium

Vulnerabilities with this level of severity can result in some damage to the protocol or users, without profit for the attacker. They often allow an attacker to exploit a contract to cause harm, but the impact may be limited, such as temporarily blocking the functionality of the protocol. Examples include uninitialized storage pointers and failure to check external calls.

Low

Vulnerabilities with this level of severity may not result in financial losses or significant harm. They may, however, impact the usability or reliability of a contract. Examples include slippage and front-running, or minor logic bugs.

Informational

Vulnerabilities with this level of severity are regarding gas optimizations and code style. They often involve issues with documentation, incorrect usage of EIP standards, best practices for saving gas, or the overall design of a contract. Examples include not conforming to ERC20, or disagreement between documentation and code.

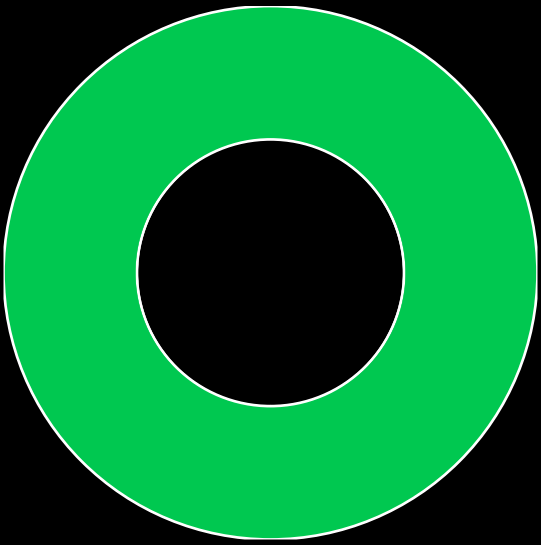
ISSUE SYMBOLIC CODES

Every issue being identified and validated has its unique symbolic code assigned to the issue at the security research stage. Cause of the vulnerability reporting flow design, some of the rejected issues could be missing.

FINDINGS SUMMARY

Severity	Number of Findings
Critical	0
High	0
Medium	0
Low	1
Informational	0

Total: 1



WEAKNESSES

This section contains the list of discovered weaknesses.

AP1-1

POTENTIAL UNDERFLOW IN FEETAKER CONTRACT

SEVERITY: Low

PATH:

contracts/extensions/FeeTaker.sol:L26-L51

REMEDIATION:

Add a check to ensure that the fee percentage obtained from the extraData parameter does not exceed the limit of 1e7 or remove the unchecked keyword from the subtraction operation takingAmount - fee to enable automatic underflow checks by Solidity.

STATUS:

DESCRIPTION:

The **FeeTaker.sol** contract contains a potential risk of underflow in the calculation of fees if the fee percentage exceeds the limit of **1e7**. In the **FeeTaker::postInteraction()** function, the fee is calculated by multiplying **takingAmount** with the fee percentage obtained from the **extraData** parameter. However, there is no explicit check to ensure that the fee percentage does not exceed the limit of **1e7**.

```
        unchecked {
            IERC20(order.takerAsset.get()).safeTransfer(receiver, takingAmount -
fee);
        }
```

Furthermore, the subtraction operation **takingAmount - fee** is currently marked with the **unchecked** keyword, which disables overflow and underflow checks. While this keyword is commonly used to optimize gas costs, it can pose a risk if the subtraction operation results in an underflow.

```
    function postInteraction(
        IOrderMixin.Order calldata order,
        bytes calldata /* extension */,
        bytes32 /* orderHash */,
        address /* taker */,
        uint256 /* makingAmount */,
        uint256 takingAmount,
        uint256 /* remainingMakingAmount */,
        bytes calldata extraData
    ) external {
        uint256 fee = takingAmount * uint256(uint24(bytes3(extraData))) /
_FEE_BASE;
        address feeRecipient = address(bytes20(extraData[3:23]));

        address receiver = order.maker.get();
        if (extraData.length > 23) {
            receiver = address(bytes20(extraData[23:43]));
        }

        if (fee > 0) {
            IERC20(order.takerAsset.get()).safeTransfer(feeRecipient, fee);
        }

        unchecked {
            IERC20(order.takerAsset.get()).safeTransfer(receiver,
takingAmount - fee);
        }
    }
```

hexens × 1inch NETWORK