Nov.23

SECURITY REVIEW REPORT FOR 1INCH

CONTENTS

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.

AUDIT LED BY

KASPER ZWIJSEN

Head of Smart Contract Audits | Hexens

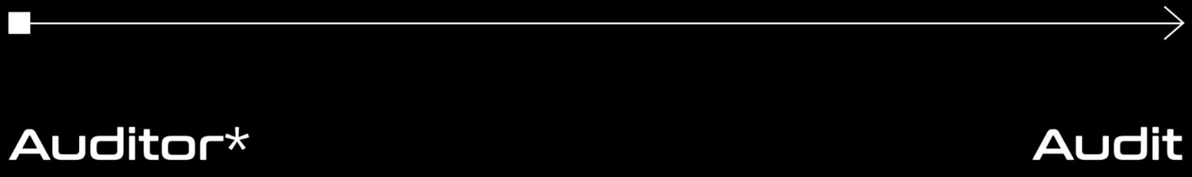
Audit Starting Date
21.11.2023

Audit Completion Date
28.11.2023

METHODOLOGY

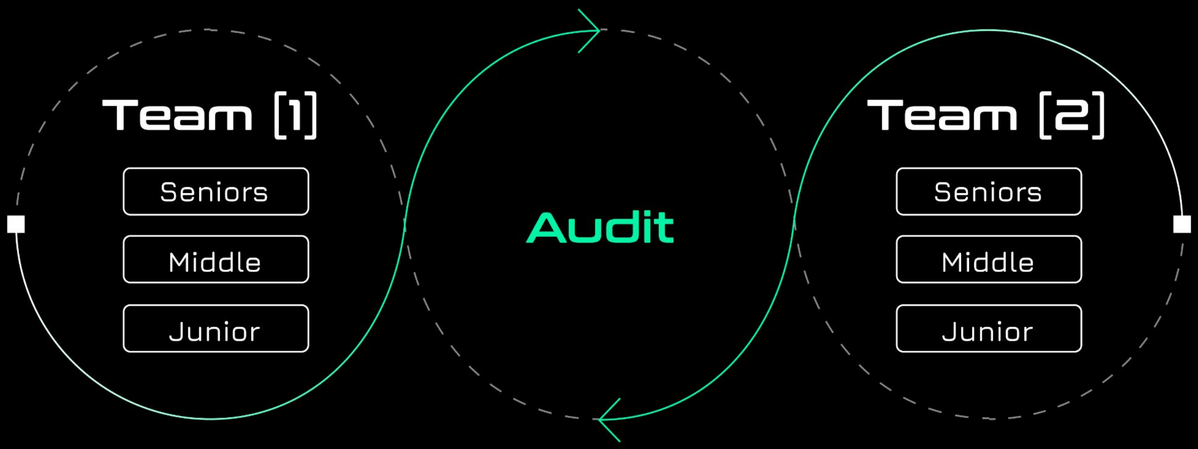
COMMON AUDIT PROCESS

Companies often assign just one engineer to one security assessment with no specified level. Despite the possible impeccable skills of the assigned engineer, it carries risks of the human factor that can affect the product's lifecycle.



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

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 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.

It's important to consider all types of vulnerabilities, including informational ones, when assessing the security of the project. A comprehensive security audit should consider all types of vulnerabilities to ensure the highest level of security and reliability.

EXECUTIVE SUMMARY

OVERVIEW

This audit covered an update to the Limit Order Protocol of 1inch Network.

Our security assessment was a full review of the new smart contract, spanning a total of 1 week.

During our audit, we have identified 1 medium severity vulnerability, which could result in a taker paying more than anticipated using a carefully constructed maker order.

The reported issue was acknowledged by the development team and marked as user risk.

SCOPE

The analyzed resources are located on:
https://github.com/1inch/limit-order-settlement/commit/ff7909c4f32bee88792116O7275dc30d788afee8

SUMMARY

SEVERITY	NUMBER OF FINDINGS
CRITICAL	0
HIGH	0
MEDIUM	1
LOW	0
INFORMATIONAL	0

TOTAL: 1

SEVERITY

STATUS

WEAKNESSES

This section contains the list of discovered weaknesses.

ONN-1. MALICIOUS MAKER CAN SIPHON THE EXCESS OF TAKER'S APPROVED TOKENS

SEVERITY: Medium

PATH: SettlementExtension.sol

REMEDIATION: the _parseFeeData() should limit the maximum fees the taker pays

STATUS: acknowledged, see commentary

DESCRIPTION:

A malicious maker can trick a benign taker into paying an exorbitant price for filling the order and bypass **takerTraits.threshold()** protection.

The **_parseFeeData()** function of **SettlementExtension** calculates **integrationFee**, which is further deducted from the taker inside **postInteraction()**. The value of **integrator** address comes directly from **extraData**. While the **integrationFee** value is calculated as follows.

```
integrationFee = actualTakingAmount * uint256(uint32(bytes4(extraData[20:24]))) / _TAKING_FEE_BASE;
```

Where 4-byte **extraData[20:24]** comes from **extraData** and may contain a maximum value up to **4.3** times bigger than **_TAKING_FEE_BASE**.

```
uint256 private constant _TAKING_FEE_BASE = 1e9;
```

The **postInteraction()** function is called as the last step of the order filling and aims for the maker to handle funds interactively.

Where **extraData** comes from the **extension** of the order (**extension.postInteractionTargetAndData()**), and the maker fully controls extension calling parameters.

- https://github.com/1inch/limit-order-protocol/blob/master/contracts/OrderMixin.sol#L291
- https://github.com/1inch/limit-order-protocol/blob/master/contracts/OrderLib.sol#L157

The attack scenario:

1. The benign taker makes an excessive approval to the **SettlementExtension** contract.
2. The malicious maker creates an order with extension parameters that transfers an excessive amount of approved **order.takerAsset.get()** tokens to the address controlled by the maker.
3. The malicious maker tricks the taker to fill the order.
4. Although **takerTraits.threshold()** is specified, the taker loses their tokens since **threshold** protection isn't applied for the post-interaction step.

Commentary from the client:

" - Won't fix. As resolvers are professional traders they expect to properly validate orders including the fee part as well. Also we don't feel that severity of the issue is High, because all the order parameters are known beforehand, so resolver can properly calculate what amount they will be charged upon filling the order."

hexens