1inch NETWORK

1inch Gasless Swaps Security Analysis

by Pessimistic

This report is public

January 27, 2023

Abstract

In this report, we consider the security of smart contracts of 1inch Gasless swaps project. Our task is to find and describe security issues in the smart contracts of the platform.

Disclaimer

The audit does not give any warranties on the security of the code. A single audit cannot be considered enough. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts. Besides, a security audit is not investment advice.

Summary

In this report, we considered the security of 1inch Gasless Swaps smart contracts. We described the audit process in the section below.

The audit showed only several issues of low severity. They do not endanger project security. Note that Staking and Stake delegation projects will partially reuse this codebase.

However, some contracts specific to Gasless Swaps project raise our concerns:

- **Settlement** contract allows arbitrary calls, includes overcomplicated interactions with limit orders, and heavily relies upon the backend to supply correct data.
- Authentication logic from **WhitelistChecker** contract unconventionally relies on `tx.origin`, and is overall non-uniform and complicates security analysis of the system.

> *Authentication logic was redesigned in the latest version of the code.*

After the initial audit, the developers fixed some issues and updated the codebase. The implementation became simpler and cleaner. However, **Settlement** contract expects very complex data as an argument. We recommend thoroughly documenting its parameters structure.

Later, the developers presented us the update #2. In this update, they fixed some issues, slightly updated the logic, and added new tests.

General recommendations

We recommend fixing the mentioned issues, simplifying interactions, and specifying the exact parameters structure of the **Settlement** contract.

Project overview

Project description

For the audit, we were provided with three projects on GitHub:

1. public 1inch ERC20 pods repository, commit 50b192a7e63d4eb4eaa8c8af431feb6113e620fd.
2. public 1inch delegating repository, commit c8f42022216458ba0425fae5c710c79d9ea8bb71.
3. private 1inch limit order settlement repository, commit 117fac4b93ce21846ef78a8554c4fcd2524481df.

The scope of the audit included all contracts from these repositories.

The documentation for the project included the document on Notion.

The total LOC of audited sources is 1254 across three projects. All 157 tests pass, and code coverage is 92%.

Codebase update #1

After the initial audit, the developers updated the codebase. For the recheck, we were provided with the following commits:

1. public 1inch ERC20 pods repository, commit fc7ff90c4c0bf567f30785fba8b1d503aa0af8b9.
2. public 1inch delegating repository, commit f9acf8191e3906cb00f7c7760eec6077523c68ff.
3. private 1inch limit order settlement repository, commit 5343f175dcd73eda3a3c0ae6357f339cf4ed6722.

The developers applied fixes and modified contract structure and logic in these updates.

Across three projects, all 184 tests pass, and code coverage is 86.3%.

Codebase update #2

After the recheck #1, the developers provided us with the update #2. This update included the following commits:

1. public 1inch ERC20 pods repository, commit 4098798e36f272804b47b7db1aa27aa3f545c9df.

2. public 1inch delegating repository, commit da8872a034a2af16fa55f88f3953689a43678844.

3. public 1inch limit order settlement repository, commit d738a07f4b131344ce0320e8e9d1679b72483379.

The update #2 included several fixes and minor logic updates. All 185 tests pass successfully, the overall code coverage is 89.4%.

Audit process

We received the complete codebase on November 3 and finished the audit on November 14, 2022.

We inspected the materials provided for the audit. We manually analyzed all the contracts within the scope of the audit and checked their logic. Among other, we verified the following properties of the code within the scope:

- Multiple contracts can be safely reused for Staking, Stake delegation, and third-party projects
- **Settlement** contract does not violate the security of limit orders from 1inch Limit Order Protocol
- Tokens cannot be withdrawn from **FeeBank** contract to unintended addresses

We scanned the project with the static analyzer Slither with our own set of rules and then manually verified all the occurrences found by the tool.

We ran tests and calculated the code coverage.

We combined in a private report all the verified issues we found during the manual audit or discovered by automated tools.

On December 22, the developers provided us with an updated version of the code. In this update, they refactored some contracts, fixed many issues from our report, and introduced multiple improvements.

We re-ran tests and Slither and manually reviewed the updated codebase. In the process, we discovered several new issues of low severity. Finally, we prepared a public report.

On January 17, 2023, the developers notified us about another update of the codebase. In this update, they fixed some issues and slightly changed the logic of the contracts. We reviewed the changes and verified the fixes. However, during the recheck, we also discovered new issues. After the review, we updated the report.

Manual analysis

The contracts were completely manually analyzed, their logic was checked. Besides, the results of the automated analysis were manually verified. All the confirmed issues are described below.

Critical issues

Critical issues seriously endanger project security. They can lead to loss of funds or other catastrophic consequences. The contracts should not be deployed before these issues are fixed.

The audit showed no critical issues.

Medium severity issues

Medium issues can influence project operation in the current implementation. Bugs, loss of potential income, and other non-critical failures fall into this category, as well as potential problems related to incorrect system management. We highly recommend addressing them.

M01. Misleading view function (new)

In the **St1inch** contract, the `earlyWithdrawLoss` function determines whether users can withdraw their stake right now and their losses. However, when the function checks if a user can perform an early withdrawal, it does not account for minimal lock period.

Low severity issues

Low severity issues do not directly affect project operation. However, they might lead to various problems in future versions of the code. We recommend fixing them or explaining why the team has chosen a particular option.

L01. CEI pattern violation (fixed)

ERC20Pods contract includes an unnecessary CEI violation in `_removeAllPods` function. We recommend removing the pod before calling `updateBalances` callback.

The issue has been fixed and is not present in the latest version of the code.

L02. Code quality (fixed)

Currently, **Pod** contract is `abstract` and contains very little logic. Consider moving it to `delegating` repo (e.g., inlining into **BasicDelegationPod** contract) or adding a minimal `updateBalances` implementation (i.e., it should check `msg.sender`).

The issue has been fixed and is not present in the latest version of the code.

L03. Inheritance order (fixed)

Consider setting `Ownable` as the first element of the inheritance list, as it is less likely to change and break storage layout of **DelegatedShare** contract.

The issue has been fixed and is not present in the latest version of the code.

L04. Excessive inheritance (resolved)

Consider moving **WhitelistChecker** logic into **Settlement** contract, since inheritance does not improve modularity in this case. The selected approach suffers from cohesion/coupling problems and subjectively complicates code structure.

Authentication logic was refactored, which resolved the issue.

L05. Memory safe assembly (fixed)

An assembly block at lines 103–111 of **ERC20Pods** contract is memory-safe. We recommend marking it accordingly.

The issue has been fixed and is not present in the latest version of the code.

L06. Gas consumption (fixed)

Consider performing a custom internal transfer instead of `_burn` and `_mint` sequence in `updateBalances` function of **BasicDelegationPod** contract. It does not touch the `totalSupply` storage slot and may emit more appropriate events.

The issue has been fixed and is not present in the latest version of the code.

L07. On-chain sorting (fixed)

Consider passing a hint to the `setMaxWhitelisted` function of **WhitelistRegistry** contract instead of semi-sorting the whitelist. E.g., provide indices that should be removed and check if it is the correct solution in O(n) time. Besides, it will significantly simplify the code.

The issue has been fixed and is not present in the latest version of the code.

L08. Code logic (resolved)

`MAX_LOCK_PERIOD` constant at line 27 of **St1inch** contract does not account for a leap year.

The codebase was updated, and this issue lost its relevance.

L09. Code quality (resolved)

In the `getFlag` function of the **AddressLib** contract, consider checking the flag as `X & flag == flag`. It is more robust and works correctly with multiple raised bits.

> *At first, devs updated the code to fix this issue, but later they removed the entire contract. Instead, the project utilizes the **AddressLib** contract from **@1inch/solidity-utils** dependency, which is technically out of the scope. Note that the initial issue is present in this particular version of **AddressLib** contract.*

L10. Code quality (fixed)

In the `_earlyWithdrawLoss` function of the **VotingPowerCalculator** contract, the scaling parameter (10/9) at line 174 depends on `_VOTING_POWER_DIVIDER`, which has not been finalized yet. Consider deriving the scaling parameter from `_VOTING_POWER_DIVIDER` or describing this dependency in a comment.

The issue has been fixed and is not present in the latest version of the code.

L11. Unindexed events (commented)

In the **IERC20Pods** contract, consider marking some event parameters as `indexed` at lines 8–9.

> *Comment from the developers: Will not fix.*

L12. Short types (addressed)

In the `ratio` function of the **TakingFee** contract, the ratio value should fit in `uint32`. However, it is never checked within the library, which might lead to an underflow. Note that the underflow cannot happen if the user forms the calldata correctly. Nonetheless, we recommend checking boundaries explicitly or removing `uint32` restriction.

L13. Misleading comment (fixed)

In the **OrderSuffix** contract, the comment at line 15 is misleading: the auction point delay takes two bytes rather than one, i.e., the comment should be: `// M*(2 + 3 bytes) ...`

The issue has been fixed and is not present in the latest version of the code.

L14. Inconsistent inheritance (new)

In the **FarmingDelegationPod** contract, the `register` function has no return value. However, the contract inherits from **TokenizedDelegationPod** and implements **ITokenizedDelegationPod** interface, and both of them include `register` function that returns `DelegatedShare` address.

L15. Fragile code (new)

In the **St1inch** contract, the `_withdraw` function leaves the `depositor.lockTime` field unchanged. It does not lead to any problems due to the details of the `_deposit` function implementation. However, it makes the code more fragile, and minor changes in the deposit logic might produce critical issues.

Notes

N01. Arbitrary calls (commented)

Settlement contract allows unrestricted calls to arbitrary targets and thus should never hold any assets or ERC20 approves from accounts with those assets.

> *Comment from the developers:* *Noted.*

N02. Limited reusability (commented)

BasicDelegationPod and **RewardableDelegationPod** contracts rely on `msg.sender`, which might limit their reusability.

> *Comment from the developers:* *Noted.*

N03. Complicated pods list tracking (fixed)

All add/remove methods of **ERC20Pods** contract skip callback if the user has zero balance. Other projects may have difficulties tracking users' pods if they change their pods list without tokens.

The issue has been fixed and is not present in the latest version of the code.

N04. lockTime reset (new)

In the `_deposit` function of the **St1inch** contract, extending the deposit duration resets the `depositor.lockTime` value. As a result, early withdrawals might become unavailable even for the smallest duration changes.

This analysis was performed by Pessimistic:

Evgeny Marchenko, Senior Security Engineer
Nikita Kirillov, Junior Security Engineer
Yhtyyar Sahatov, Junior Security Engineer
Boris Nikashin, Analyst

January 27, 2023