·Decurity·

Smart Contract Security Audit

Re-Test Report

1inch Fusion

Decurity, 2022

Contents

1. General Information

This report contains information about the results of the security audit of the 1inch (hereafter referred to as “Customer”) smart contracts, conducted by Decurity in the period from 12/01/2022 to 12/26/2022.

1.1. Introduction

Tasks solved during the work are:

- Review the protocol design and the usage of 3rd party dependencies,
- Audit the contracts implementation,
- Develop the recommendations and suggestions to improve the security of the contracts.

1.2. Scope of Work

The audit scope included the contracts in the following repositories:

- https://github.com/1inch/limit-order-settlement (commits fb55a16b19d00c4bde310b276bfeecf23116c371, 0e8189419c5fbbc67416a1d9e7721cb06b51a38b, c90dae1a89cd493aeed33331f86e2f3393aea95f),
- https://github.com/1inch/erc20-pods/ (commit 2b4545d435e159f08ddf80fd14e8f4efb665bcdb),
- https://github.com/1inch/delegating (commits 9a243d64422c41b0631617466deeb85dcd92e57c, a7fe161c6c94eb6d2d4cd8a735ef150d1133d7db)
- https://github.com/1inch/farming (commits 8403ce63df903db66c5c8fd7958470c6827df7e6, 08e99d163429f2645e28f2eccb1345dc405861cd)

1.3. Threat Model

The assessment presumes actions of an intruder who might have capabilities of any role (an external user, token owner, a contract). The centralization risks have not been considered.

The main possible threat actors are:

- User,
- Protocol owner,
- Limit Order Protocol contract,
- Settlement resolver (whitelisted contract),
- 1inch backend (the source of the orders).

The table below contains sample attacks that malicious attackers might carry out.

Table. Theoretically possible attacks

Attack	Actor
Contract code or data hijacking *Deploying a malicious contract or submitting malicious data*	Protocol owner
Financial fraud *A malicious manipulation of the business logic and balances, such as a re-entrancy attack or a flash loan attack*	Anyone
Attacks on implementation *Exploiting the weaknesses in the compiler or the runtime of the smart contracts*	Anyone

1.4. Weakness Scoring

An expert evaluation scores the findings in this report, an impact of each vulnerability is calculated based on its ease of exploitation (based on the industry practice and our experience) and severity (for the considered threats).

2. Summary

As a result of this work, we've described several suggestions included fixing the low-risk issues and some best practices (see 3.1).

The 1inch team has given the feedback for the suggested changes and explanation for the underlying code.

2.1. Suggestions

The table below contains the discovered issues, their risk level, and their status as of Jun 15, 2022.

Table. Discovered weaknesses

Issue	Contract	Risk Level	Status
St1inch _withdraw argument inconsistency	St1inch.sol	Low	Fixed
Missing 0x0 check on input addresses	St1inch.sol	Low	Fixed
Early withdraw can revert	St1inch.sol	Low	Not Fixed
If the deposit lock expires in less than 30 days, rewards from StakingFarmingPod cannot be claimed	StakingFarmingPod.sol	Low	Acknowledged (not a bug)
Non-optimal addition/subtraction	St1inch.sol FeeBank.sol	Info	Fixed (mostly)
Unused VotingPowerCalculator import	WhitelistRegistry.sol	Info	Fixed

Unnecessary checked arithmetic	WhitelistRegistry.sol FeeBank.sol	**Info**	Not Fixed
Non-optimal increment/decrement	WhitelistRegistry.sol FeeBank.sol	**Info**	Fixed
FeeBank should inherit from IFeeBank	FeeBank.sol	**Info**	Fixed
Unused library functions	OrderSaltParser.sol Address.sol	**Info**	Not Fixed
Split if statement that using &&	St1inch.sol ERC20Pods.sol	**Info**	Not Fixed
A user can be unwillingly added to the Pod	St1inch.sol	**Info**	Not Fixed
Constant _ONE is overridden	VotingPowerCalculator.sol St1inch.sol	**Info**	Not Fixed
Variables shadowing	FeeBank.sol DelegatedShare.sol FarmingDelegationPod.sol FarmingPod.sol TokenizedDelegationPod.sol MultiFarmingPod.sol	**Info**	Not Fixed

3. General Recommendations

This section contains general recommendations on how to fix discovered weaknesses and vulnerabilities and how to improve overall security level.

Section 3.1 contains a list of general mitigations against the discovered weaknesses, technical recommendations for each finding can be found in section 4.

Section 3.2 describes a brief long-term action plan to mitigate further weaknesses and bring the product security to a higher level.

3.1. Current Findings Remediation

Follow the recommendations in section 4.

3.2. Security Process Improvement

- Keep the whitepaper and documentation updated to make it consistent with the implementation and the intended use cases of the system,
- Perform regular audits for all the new contracts and updates,
- Ensure the secure off-chain storage and processing of the credentials (e.g. the privileged private keys),
- Launch a public bug bounty campaign for the contracts.

4. Findings

4.1. St1inch _withdraw argument inconsistency

Risk Level: Low

Contracts:

- St1inch.sol

Contracts: Fixed in the following commit: https://github.com/1inch/limit-order-settlement/commit/900327ebc58f38ad6442c922d2d0dd47484926cd.

Description:

The function `_withdraw` accepts the argument `amount` separately from the argument `depositor`. However, the `depositor` value already contains the `amount` value in the corresponding structure field.

This may lead to an incorrect usage of the function in the future if it's called with desynchronized values (e.g. `amount` differs from `depositor.amount`).

Remediation:

Use `depositor.amount` instead of `amount`.

4.2. Missing 0x0 check on input addresses

Risk Level: Low

Contracts:

- St1inch.sol

Status: Fixed in the following commit: https://github.com/1inch/limit-order-settlement/commit/39d568da8574f5891bbdf1962fe50f988969643a

Description:

There are multiple occurrences of dangerous assignment of the input address to a storage variable without proper sanitization, namely not checking that the address is zero.

There are the following occurrences:

- Variable feeReceiver_ in setFeeReceiver() function of the St1inch.sol

Remediation:

Always check that an input address is not zero before assigning a storage variable.

4.3. Early withdraw can revert

Risk Level: Low

Contracts:

- St1inch.sol

Description:

Early withdraw can revert in two cases:

- the amount is very small (hundreds or thousands of decimals),
- an attempt to withdraw occurs close to the unlockTime point.

Below is the test code that demonstrates the attack:

```
     it('early withdraw', async function () {
         const { st1inch } = await loadFixture(initContracts);

                        await    st1inch.deposit(ether('0.0000000001'),
time.duration.days('310'));
                                          unlock        =        (await
st1inch.depositors(addr.address)).unlockTime;
         await timeIncreaseTo(unlock - 1);
                                     console.log('loss:',        await
st1inch.earlyWithdrawLoss(addr.address))
```

```
});
```

Remediation:

Add a warning to the frontend if a user tries to deposit a very small amount or tries to withdraw around the unlock time.

4.4. If the deposit lock expires in less than 30 days, rewards from StakingFarmingPod cannot be claimed

Risk Level: Low

Contracts:

- StakingFarmingPod.sol

Status: Acknowledged: this is an intended staking incentivization mechanic.

Description:

Because the _transferReward() method in the StakingFarmingPod contract calls the depositFor() of the St1inch, which calls the _deposit() function with duration equal to 0, rewards from StakingFarmingPod cannot be claimed if the deposit lock expires in less than 30 days.

Users will not be able to collect rewards if the deposit's lockLeft is shorter than 30 days, and they must adjust the length of their deposits in order to do so.

Below is the test code that demonstrates the attack:

```
it('staking pod', async function () {
      const { oneInch, st1inch, stakingfarmingpod } = await
loadFixture(initContracts);
```

```
        await st1inch.addPod(stakingfarmingpod.address);

        await stakingfarmingpod.setDistributor(addr.address);
                  await   oneInch.approve(stakingfarmingpod.address,
'1000000000');
         const started = await startFarming(stakingfarmingpod, 1000,
60 * 60 * 24, addr);

                              await      st1inch.deposit(ether('10'),
time.duration.days('31'));
                                       unlock        =        (await
st1inch.depositors(addr.address)).unlockTime;
        await timeIncreaseTo(unlock);
        await stakingfarmingpod.claim();
     });
```

Remediation:

Consider transferring rewards to a user instead of depositing it for them.

4.5. Non-optimal addition/subtraction

Risk Level: Info

Contracts:

- St1inch.sol
- FeeBank.sol

References:

- https://gist.github.com/llllllll000/cbbfb267425b898e5be734d4008d4fe8

Status: Partially fixed in the following commit: https://github.com/1inch/limit-order-settlement/commit/5b513b92b357d9419cfeae42a9b668e714beb495.

Description:

Using the addition/submission operator instead of plus-equals/minus-equals saves gas.

There are the following cases:

- contracts/St1inch.sol:
  - 123: totalDeposits += amount;
  - 175: totalDeposits -= amount;
- contracts/FeeBank.sol:
  - 98: _accountDeposits[accounts[i]] -= accountFee;
  - 106: _accountDeposits[account] += amount;
  - 112: _accountDeposits[msg.sender] -= amount;

Remediation:

Use `depositor.amount` instead of `amount`.

This an example of a not optimized code:

```
issuedSupply += amount;
```

Consider using addition/submission operators:

```
issuedSupply = issuedSupply + amount;
```

4.6. Unused VotingPowerCalculator import

Risk Level: Info

Contracts:

- WhitelistRegistry.sol

Status: Fixed in the following commit: https://github.com/1inch/limit-order-settlement/commit/6e6cf71c9eba47f1bee24e86b7a3edd61e00c57b.

Description:

There is unused import of VotingPowerCalculator.sol in WhitelistRegistry.sol#9.

Remediation:

Remove unnecessary imports from the code to prevent confusion.

4.7. Unnecessary checked arithmetic

Risk Level: Info

Contracts:

- WhitelistRegistry.sol
- FeeBank.sol

Description:

Use `unchecked` blocks where overflow/underflow is impossible. There are several cycles where arithmetic checks are not necessary:

- WhitelistRegistry.sol#L68
- WhitelistRegistry.sol#L122
- WhitelistRegistry.sol#L129
- WhitelistRegistry.sol#L137
- WhitelistRegistry.sol#L146
- FeeBank.sol#96

Remediation:

Consider using unchecked to save gas.

4.8. Non-optimal increment/decrement

Risk Level: Info

Contracts:

- WhitelistRegistry.sol
- FeeBank.sol

Status: Fixed in the following commit: https://github.com/1inch/limit-order-settlement/commit/be7d2aae6790d0c838c5db06a7211d472daaab09.

Description:

The difference between the prefix increment and postfix increment expression lies in the return value of the expression.

The prefix increment expression (++i) returns the *updated* value after it's incremented. The postfix increment expression (i++) returns the *original* value.

The prefix increment expression is cheaper in terms of gas.

Consider using the prefix increment expression whenever the return value is not needed.

There are the following occurrences:

- WhitelistRegistry.sol#L97
- WhitelistRegistry.sol#L122
- WhitelistRegistry.sol#L129
- WhitelistRegistry.sol#L137
- WhitelistRegistry.sol#L146
- FeeBank.sol#96

Remediation:

Consider using pre-increment to save gas.

4.9. FeeBank should inherit from IFeeBank

Risk Level: Info

Contracts:

- FeeBank.sol

Status: Fixed in the following commit: https://github.com/1inch/limit-order-settlement/commit/babc5210dfcdde5f5afa1c26b9589b254b7268b8.

Description:

The FeeBank contract doesn't inherit from the IFeeBank interface.

Remediation:

Inherit FeeBank contract from the IFeeBank interface.

4.10. Unused library functions

Risk Level: Info

Contracts:

- OrderSaltParser.sol
- Address.sol

Description:

There are functions that are never used:

- OrderSaltParser.getSalt(uint256) (contracts/libraries/OrderSaltParser#34-36)
- AddressLib.getUint64(Address,uint256) (contracts/libraries/Address.sol#20-22)

Remediation:

Consider removing the unused functions from the libraries.

4.11. Split if statement that using &&

Risk Level: Info

Contracts:

- St1inch.sol
- ERC20Pods.sol

References:

- https://github.com/code-423n4/2022-01-xdefi-findings/issues/128

Description:

Gas costs are higher when using &&.:

- contracts/St1inch.sol:
  - 71: if (defaultFarm_ != address(0) && Pod(defaultFarm_).token() != address(this)) revert DefaultFarmTokenMismatch();
  - 140: if (defaultFarm != address(0) && !hasPod(account, defaultFarm)) {
  - 184: if (!emergencyExit && block.timestamp < depositor.unlockTime) revert UnlockTimeHasNotCome();
- contracts/ERC20Pods.sol:
  - 136: if (amount > 0 && from != to) {

Remediation:

Consider separate check instead of &&.

4.12. A user can be unwillingly added to the Pod

Risk Level: Info

Contracts:

- St1inch.sol

Description:

A user can be forcibly added to a pod (`defaultFarm`) even if they opted out. To do that, an attacker can deposit amount 0 for the user. The `_deposit` function will call `_addPod` for the target account.

Remediation:

If this is an undesired behaviour, at least opt to the pod only if the amount is non-zero. Also `_addPod` can be called only when a user deposits for themselves.

4.13. Constant _ONE is overridden

Risk Level: Info

Contracts:

- VotingPowerCalculator.sol
- St1inch.sol

Description:

The VotingPowerCalculator.sol file has a constant named `_ONE` that has the value `1e18`. This constant is used in the St1inch contract, which inherits from VotingPowerCalculator, and is overridden with the `1e9` value.

Remediation:

Consider renaming a value.

4.14. Variables shadowing

Risk Level: Info

Contracts:

- FeeBank.sol
- DelegatedShare.sol
- FarmingDelegationPod.sol
- FarmingPod.sol
- TokenizedDelegationPod.sol

- MultiFarmingPod.sol

Description:

The variable `owner` in the `constructor()` function shadows the state variable `owner` inherited from the Ownable contract.

There is the following occurrences:

- FeeBank.sol#L23

The `name` and `symbol` variable in the `constructor()` function shadows the state `name` and `symbol` variables inherited from the ERC20 contract.

There is the following occurrences:

- DelegatedShare.sol#L26

The `name` and `symbol` variable in the `register()` function shadows the state `name` and `symbol` variables inherited from the ERC20 contract.

There are the following occurrences:

- FarmingDelegationPod.sol#L21
- TokenizedDelegationPod.sol#L40

The `podCallGasLimit` variable in the `contsructor()` function shadows the state `podCallGasLimit` variable inherited from the ERC20Pods contract.

There is the following occurrences:

- DelegatedShare.sol#L26

The variable `token` in the `rescueFunds()` function shadows the state variable `token` inherited from the Pod contract.

There is the following occurrences:

- FarmingPod.sol#L96
- FarmingPod.sol#L99
- MultiFarmingPod.sol#L133

- MultiFarmingPod.sol#L136

Remediation:

To prevent shadowing, consider renaming the variables.

5. Appendix

5.1. About us

The Decurity (former DeFiSecurity.io) team consists of experienced hackers who have been doing application security assessments and penetration testing for over a decade.

During the recent years, we've gained expertise in the blockchain field and have conducted numerous audits for both centralized and decentralized projects: exchanges, protocols, and blockchain nodes.

Our efforts have helped to protect hundreds of millions of dollars and make web3 a safer place.