Smart Contract Security Audit Report

1inch

1. Contents

2. 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 24/02/2025 to 28/02/2025.

2.1. 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.

2.2. Scope of Work

The audit scope included the contracts in the following repository: solana-fusion (commit a52507d). Retesting was done for the commit fef0cde.

The following contracts have been tested:

- /programs/**/*.*

2.3. Threat Model

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

The main possible threat actors are:

- Protocol Owner
- Taker
- Maker

2.4. 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).

2.5. Disclaimer

Due to the intrinsic nature of the software and vulnerabilities and the changing threat landscape, it cannot be generally guaranteed that a certain security property of a program holds.

Therefore, this report is provided “as is” and is not a guarantee that the analyzed system does not contain any other security weaknesses or vulnerabilities. Furthermore, this report is not an endorsement of the Customer’s project, nor is it an investment advice.

That being said, Decurity exercises best effort to perform their contractual obligations and follow the industry methodologies to discover as many weaknesses as possible and maximize the audit coverage using the limited resources.

3. Summary

During the audit, we detected multiple low and informational issues.

3.1. Suggestions

The table below contains the discovered issues, their risk level, and their status as of March 14, 2025.

Table. Discovered weaknesses

Issue	Contract	Risk Level	Status
Different fee calculation method could lead to DoS	programs/fusion-swap/src/lib.rs	Low	Acknowledged
Maker can avoid paying the fees	programs/fusion-swap/src/lib.rs	Low	Acknowledged
Use two-step ownership transfer	programs/whitelist/src/lib.rs	Low	Acknowledged
Whitelist initializer can be front-run	solana-fusion/programs/whitelist/src/lib.rs	Low	Acknowledged
Inconsistency with the documentation	solana-fusion/programs/fusion-swap/src/lib.rs	Info	Fixed
Fee recipients are not validated	programs/fusion-swap/src/lib.rs	Info	Acknowledged

4. General Recommendations

This section contains general recommendations on how to improve overall security level.

The Findings section contains technical recommendations for each discovered issue.

4.1. Security Process Improvement

The following is a brief long-term action plan to mitigate further weaknesses and bring the product security to a higher level:

- 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.

5. Findings

5.1. Different fee calculation method could lead to DoS

Risk Level: Low

Status: Client's comment: Noted. Users will always have various ways to create orders that will never be filled, potentially enabling spam attacks. However, this attack would not impact resolvers, as our backend validates all orders. If an order contains invalid data, it will not be forwarded to resolvers. Additionally, this type of attack is quite costly since each order requires at least two irreversible transactions (Create and Cancel), making it impractical in real-world scenarios.

Contracts:

- programs/fusion-swap/src/lib.rs

Location: Function: `create`.

Description:

In Solana Fusion, when both the integrator fee and protocol fee are set to 50%, the total fee sums up to 100%.

However, in the EVM version, due to a different calculation method, the total fee amounts to only 50% (due to the denominator).

This discrepancy can cause issues in the Solana implementation when the total fee exceeds 100%, making it impossible to fill the order due to an underflow error.

Remediation:

Ensure that `integrator fee + protocol fee < BASE_1E5` during order creation to prevent this issue.

5.2. Maker can avoid paying the fees

Risk Level: Low

Status: Client's comment: Noted. Similar to 5.1, orders with invalid data will be rejected by our backend and will not be filled. Consequently, once fees are enabled, any orders that do not include a fee will be considered invalid by the backend.

Contracts:

- programs/fusion-swap/src/lib.rs

Location: Function: `get_fee_amounts`.

Description:

The order maker has full control over protocol fees and can bypass them entirely by excluding fee settings from the order configuration.

Remediation:

To prevent fee config manipulation, consider enforcing a minimum fixed fee for order.

5.3. Use two-step ownership transfer

Risk Level: Low

Status: Client's comment: Noted. In the unlikely event that control over the whitelist contract is lost in this way, we would need to redeploy the contract at a new address. However, this would not cause any significant disruptions to the protocol's functionality. Therefore, we do not see a strong need to introduce additional checks at this stage.

Contracts:

- programs/whitelist/src/lib.rs

Location: Function: `transfer_ownership()`.

Description:

The recommended approach for transferring ownership is to require the new owner to explicitly accept it. Only after acceptance does the new owner become valid. This prevents issues such as accidentally assigning ownership to an invalid address.

Remediation:

Require the new owner to accept the ownership after its transfer before it becomes valid.

5.4. Whitelist initializer can be front-run

Risk Level: Low

Status: Client's comment: Noted. If such an attack were attempted, we would likely detect it before the first order is created and simply redeploy the contracts with updated parameters. We can implement the validation check you suggested in the future if front-running becomes a real concern.

Contracts:

- solana-fusion/programs/whitelist/src/lib.rs

Location: Function: `initialize()`.

Description:

Since the whitelist account is created with a constant seed (`WHITELIST_STATE_SEED`), any user could front-run the intended owner's initialization transaction and become the owner of the whitelist by calling initialize first. This would give them complete control over the whitelist functionality.

Remediation:

Either hardcode the deployer address in the program and allow the `initalize()` function to be called by this address only, or send the deployment and initialization transactions as a bundle via the Jito MEV infrastructure.

5.5. Inconsistency with the documentation

Risk Level: Info

Status: Fixed in the commit fef0cde

Contracts:

- solana-fusion/programs/fusion-swap/src/lib.rs

Location: Function: `fill()`.

Description:

If a native destination asset is selected, the fee can be paid in WSOL only, but the taker amount will be transferred in native SOL. However, according to the documentation, the fee should be paid in the same asset as the taker's asset.

Remediation:

Allow fee payments in native SOL in case it is used as the taker's asset.

5.6. Fee recipients are not validated

Risk Level: Info

Status: Client's comment: As with 5.1 and 5.2, these parameters will be validated on the backend.

Contracts:

- programs/fusion-swap/src/lib.rs

Location: Function: `create`.

Description:

When creating an order, `protocol_dst_ata` and `integrator_dst_ata` are not validated, allowing the maker to provide any address they choose.

Remediation:

Consider validating this address via global config.

6. Appendix

6.1. About us

The Decurity 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.