1inch Vesting Contract

SMART CONTRACT AUDIT

25.05.2021

Made in Germany by Chainsulting.de

Table of contents

1. Disclaimer

The audit makes no statements or warrantees about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of 1Inch Exchange. If you are not the intended receptor of this document, remember that any disclosure, copying or dissemination of it is forbidden.

Major Versions / Date	Description
0.1 (17.05.2021)	Layout
0.2 (18.05.2021)	Test Deployment
0.5 (19.05.2021)	Automated Security Testing Manual Security Testing
0.7 (20.05.2021)	Verify Claims
0.9 (21.05.2021)	Summary and Recommendation
1.0 (21.05.2021)	Final document
1.1 (TBA)	Added deployed contract addresses

2. About the Project and Company

Company address:

1Inch Limited
Quijano Chambers, P.O. Box 3159, Road Town
Tortola, British Virgin Islands

Sergej Kunz Co-Founder & Chief Executive Officer
Anton Bukov Co-Founder & Chief Technology Officer

Discord: https://discord.gg/FZADkCZ

Blog: https://blog.1inch.io

Medium: https://medium.com/@1inch.exchange

Website: https://app.1inch.io

Twitter: https://twitter.com/1inchExchange

Reddit: https://www.reddit.com/r/1inch_exchange

Telegram: https://t.me/OneInchExchange

Forum: https://gov.1inch.io

2.1 Project Overview

1inch is a so-called DEX aggregator, which means that it scrapes a handful of decentralized exchanges for the cheapest prices and reroutes its customers' trades between them to try and ensure that they're getting the best prices. This is particularly important when exchanging large token amounts as it reduces the price slippage, ensuring trades are optimized for the best price.

1inch was founded by Sergej Kunz and Anton Bukov in 2019 during ETHNewYork's hackathon. Since then, 1inch has raised about $15 million in funding from companies such as Binance Labs, Galaxy Digital and Pantera Capital. As of January 2021, 1inch's exchange trades about $155 million a day.

3. Vulnerability & Risk Level

Risk represents the probability that a certain source-threat will exploit vulnerability, and the impact of that event on the organization or system. Risk Level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 – 10	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon as possible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 – 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

4. Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices. To do so, reviewed line-by-line by our team of expert pentesters and smart contract developers, documenting any issues as there were discovered.

4.1 Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
   i. Review of the specifications, sources, and instructions provided to Chainsulting to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Chainsulting describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analysing a program to determine what inputs causes each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts.

4.2 Used Code from other Frameworks/Smart Contracts (direct imports)

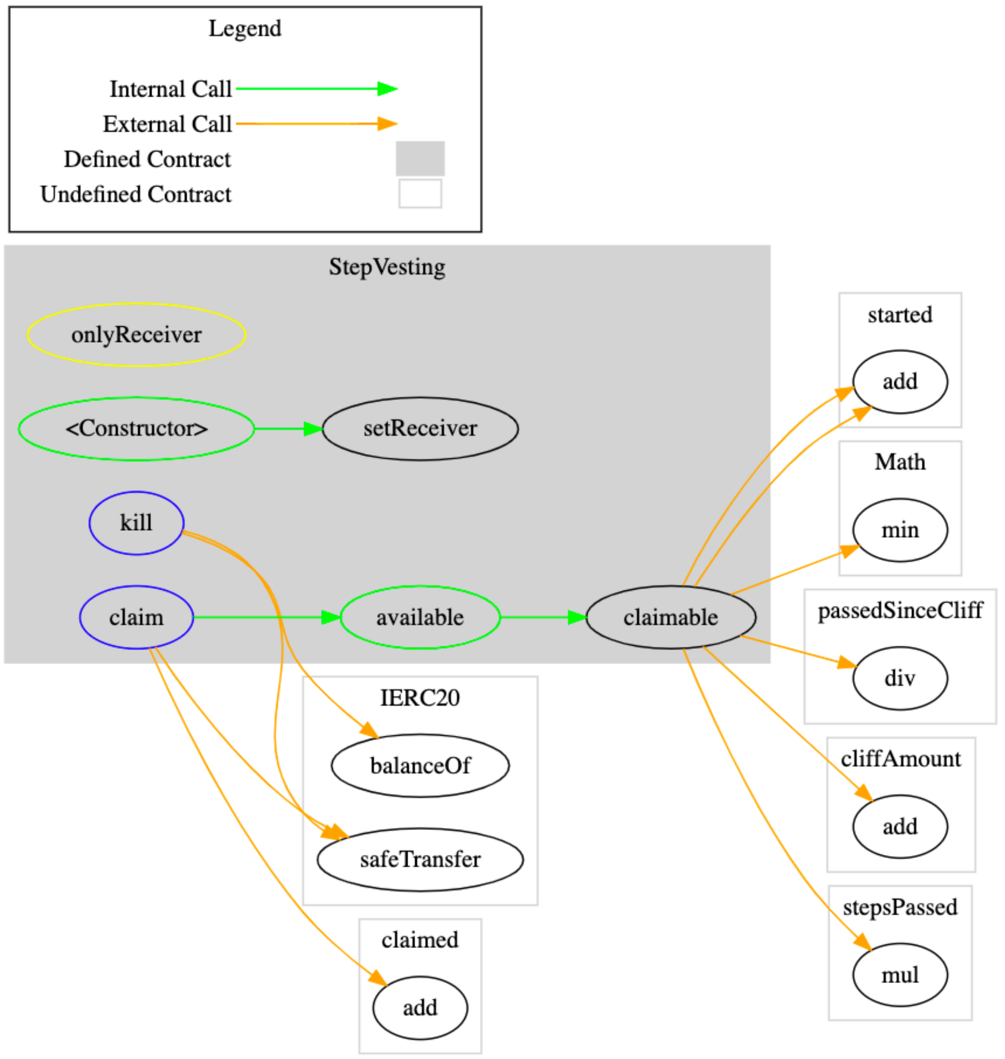
Dependency / Import Path	**Link / Source**
@openzeppelin/contracts/math/Math.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v3.0.0/contracts/math/Math.sol
@openzeppelin/contracts/math/SafeMath.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v3.0.0/contracts/math/SafeMath.sol
@openzeppelin/contracts/access/Ownable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v3.0.0/contracts/access/Ownable.sol
@openzeppelin/contracts/token/ERC20/SafeERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v3.0.0/contracts/ERC20/SafeERC20.sol

4.3 Tested Contract Files

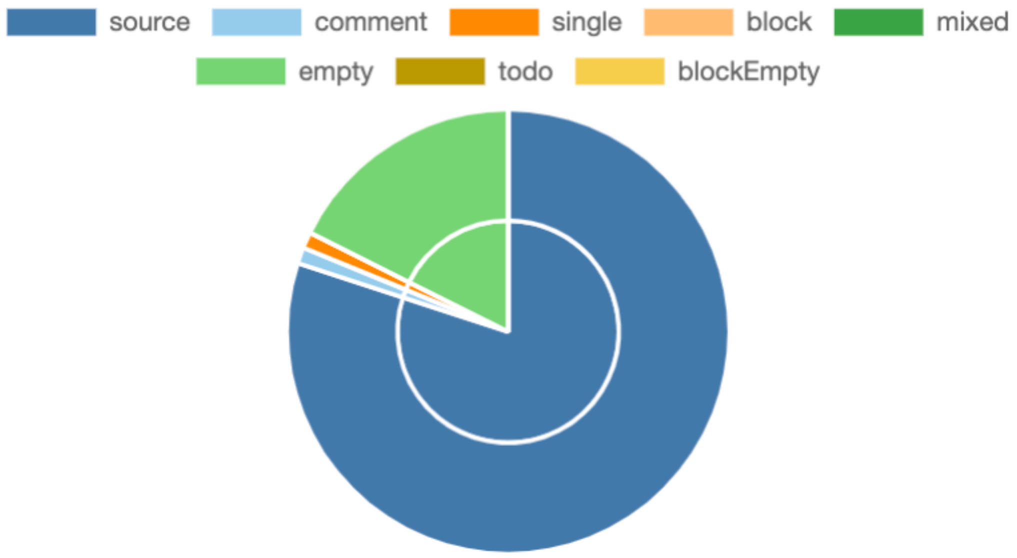
The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review

File	Fingerprint (MD5)
StepVesting.sol	6781b0d7fe378bc9c0d88fa80182304e

4.4 Metrics / CallGraph



4.5 Metrics / Source Lines



4.6 Metrics / Capabilities

Solidity Versions observed	🧪 **Experimental Features**	💰 **Can Receive Funds**	🖥 **Uses Assembly**	💣 **Has Destroyable Contracts**
`^0.6.0`			**** (0 asm blocks)	

📤 **Transfers ETH**	⚡ **Low-Level Calls**	👥 **DelegateCall**	🧮 **Uses Hash Functions**	🔖 **ECRecover**	🌀 **New/Create/Create2**

Exposed Functions
This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.

🌐**Public**	💰**Payable**
6	0

External	**Internal**	**Private**	**Pure**	**View**
2	4	0	0	2

StateVariables

Total	🌐**Public**
9	9

4.7 Metrics / Source Unites in Scope

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📝	Vesting/StepVesting.sol	1		84	84	68	1	46	
📝	**Totals**	**1**		**84**	**84**	**68**	**1**	**46**	

Legend: [➖]

- **Lines**: total lines of the source unit
- **nLines**: normalized lines of the source unit (e.g. normalizes functions spanning multiple lines)
- **nSLOC**: normalized source lines of code (only source-code lines; no comments, no blank lines)
- **Comment Lines**: lines containing single or block comments
- **Complexity Score**: a custom complexity score derived from code statements that are known to introduce code complexity (branches, loops, calls, external interfaces, ...)

5. Scope of Work

The 1inch Team provided us with the files that needs to be tested. The scope of the audit is the Step Vesting contract.

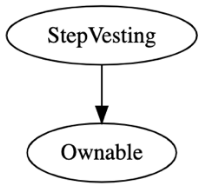
Following contracts with the direct imports been tested

- StepVesting.sol

The team put forward the following assumptions regarding the security, usage of the contracts:

- The cliff-, step-duration is correctly working
- Deployer/Owner cannot burn any vested funds during the vesting period
- Deployer/Owner cannot pause the contract
- Beneficiaries/Receiver can withdraw tokens after vesting period ends or during the step duration.
- Deployer/Owner cannot withdraw token from vesting contract
- The smart contract is coded according to the newest standards and in a secure way.

The main goal of this audit was to verify these claims. The auditors can provide additional feedback on the code upon the client's request.



5.1 Manual and Automated Vulnerability Test

CRITICAL ISSUES

During the audit, Chainsulting‘s experts found **no Critical issues** in the code of the smart contract.

HIGH ISSUES

During the audit, Chainsulting’s experts found **no High issues** in the code of the smart contract.

MEDIUM ISSUES

During the audit, Chainsulting’s experts found **no Medium issues** in the code of the smart contract.

LOW ISSUES

During the audit, Chainsulting’s experts found **no Low issues** in the code of the smart contract.

INFORMATIONAL ISSUES

5.1.1 Floating pragma

Severity: INFORMATIONAL
Status: Acknowledged
File(s) affected: StepVesting.sol

Attack / Description	Code Snippet	Result/Recommendation
The Contract should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively. Also the pragma version 0.6.0 is giving out an error after compiling: StepVesting.sol:17:20: ParserError: Expected identifier but got reserved keyword 'immutable' uint256 public immutable started; ^-------^	`pragma solidity ^0.6.0;`	Recommend fixing pragma to 0.6.12 https://consensys.github.io/smart-contract-best-practices/recommendations/#lock-pragmas-to-specific-compiler-version

5.1.2 Missing natspec documentation

Severity: INFORMATIONAL
Status: Acknowledged
File(s) affected: StepVesting.sol

Attack / Description	Code Snippet	Result/Recommendation
Solidity contracts can use a special form of comments to provide rich documentation for functions, return variables and more. This special form is named the Ethereum Natural Language Specification Format (NatSpec).	NA	It is recommended to include natspec documentation and follow the doxygen style including @author, @title, @notice, @dev, @param, @return and make it easier to review and understand your smart contract. The team addressed the issue while auditing and added more documentation parts. https://github.com/1inch-exchange/mooniswap-v2/pull/13/commits

6. Test Deployment & Verify Claims

6.1 The cliff-, step-duration is correctly working
Status: tested and verified ✅

6.2 Beneficiaries/Receiver can withdraw tokens after vesting period ends or during the step duration.
Status: tested and verified ✅

6.3 Deployer/Owner cannot burn any vested funds during the vesting period
Status: tested and verified ✅

6.4 Deployer/Owner cannot pause the contract
Status: tested and verified ✅

Deployer/Owner can change the receiver, depending on the use case, this can be used in a malicious way.

Update from Team: Change receiver is required because people are losing access to their wallets more often that they should. Kill is needed because we can legally break the contract, so we need to have that option in smart contract too. Context is that for each deployed vesting contract we have an accompanying legal agreement.

Line: 69 - 72

```
function setReceiver(address _receiver) public onlyOwner {
        emit ReceiverChanged(receiver, _receiver);
        receiver = _receiver;
    }
```

6.5 Deployer/Owner cannot withdraw token from vesting contract

Status: tested and verified ✅

Deployer/Owner can withdraw token with kill function, depending on the use case, this can be used in a malicious way.

Update from Team: Change receiver is required because people are losing access to their wallets more often that they should. Kill is needed because we can legally break the contract, so we need to have that option in smart contract too. Context is that for each deployed vesting contract we have an accompanying legal agreement.

Line: 74 - 77

```
    function kill(address target) external onlyOwner {
        uint256 amount = token.balanceOf(address(this));
        token.safeTransfer(target, amount);
    }
```

7. Executive Summary

Two (2) independent Chainsulting experts performed an unbiased and isolated audit of the smart contract codebase. The final debriefs took place on the May 21, 2021. The overall code quality of the project is good, not overloaded with unnecessary functions, these is greatly benefiting the security of the contract. It correctly implemented widely used and reviewed contracts from OpenZeppelin.

The main goal of the audit was to verify the claims regarding the security of the smart contract and the functions. During the audit, no issues were found after the manual and automated security testing and the claims been successfully verified. The only concern is the kill and change receiver function that can be used by the owner, please clarify the use case with the auditor.

Update from Team: Change receiver is required because people are losing access to their wallets more often that they should. Kill is needed because we can legally break the contract, so we need to have that option in smart contract too. Context is that for each deployed vesting contract we have an accompanying legal agreement.

8. Deployed Smart Contract

PENDING

Contract is deployed here:
https://etherscan.....