



Furucombo (COMBO Token)
SMART CONTRACT AUDIT

07.04.2021

Made in Germany by Chainsulting.de



Table of contents

1. Disclaimer	3
2. About the Project and Company	4
2.1 Project Overview	5
3. Vulnerability & Risk Level.....	6
4. Auditing Strategy and Techniques Applied	7
4.1 Methodology.....	7
4.2 Used Code from other Frameworks/Smart Contracts	8
4.3 Tested Contract Files.....	8
4.4 Metrics / CallGraph	9
4.6 Metrics / Capabilities.....	11
4.7 Metrics / Source Unites in Scope.....	12
5. Scope of Work.....	13
5.1 Manual and Automated Vulnerability Test	14
5.2. SWC Attacks & Special Checks	15
6. Verify Claims	19
7. Executive Summary	21
8. Deployed Smart Contract.....	21



1. Disclaimer

The audit makes no statements or warranties 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 Furucombo by DINNGO Pte. Ltd. . 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 (24.03.2021)	Layout
0.5 (25.03.2021)	Verify Claims and Test Deployment
0.6 (26.03.2021)	Testing SWC Checks
0.8 (26.03.2021)	Automated Security Testing Manual Security Testing
0.9 (27.03.2021)	Summary and Recommendation
1.0 (27.03.2021)	Final document



2. About the Project and Company

Company address:

DINNGO Pte. Ltd.
100 Tras Street #16-01

Singapore 079027

Website: <https://furucombo.app>

Twitter: <https://twitter.com/furucombo>

Medium: <https://medium.com/furucombo>

Telegram: <https://t.me/furucombo>

YouTube: <https://www.youtube.com/channel/UCa1kGD4lvTSrmfKbDjQNOxQ>

Discord: <https://discord.furucombo.app>

2.1 Project Overview

Furucombo is a tool built for end-users to optimize their DeFi strategy simply by drag and drop. It visualizes complex DeFi protocols into cubes. Users setup inputs/outputs and the order of the cubes, then Furucombo bundles all the cubes into one transaction and sends out. Furucombo calls this building-blocks setup a “combo”.

The allocation of COMBO ensures stable development and maintenance of Furucombo while allowing the community to lead Furucombo’s governance. The launch of COMBO Token represents a tremendous milestone for Furucombo and DeFi. It demonstrates the maturity of our product and our determination of becoming a super aggregator. Furucombo believes that a community-driven product would open up a world of infinite possibilities and they’re absolutely excited to explore the next chapter of Furucombo.



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)

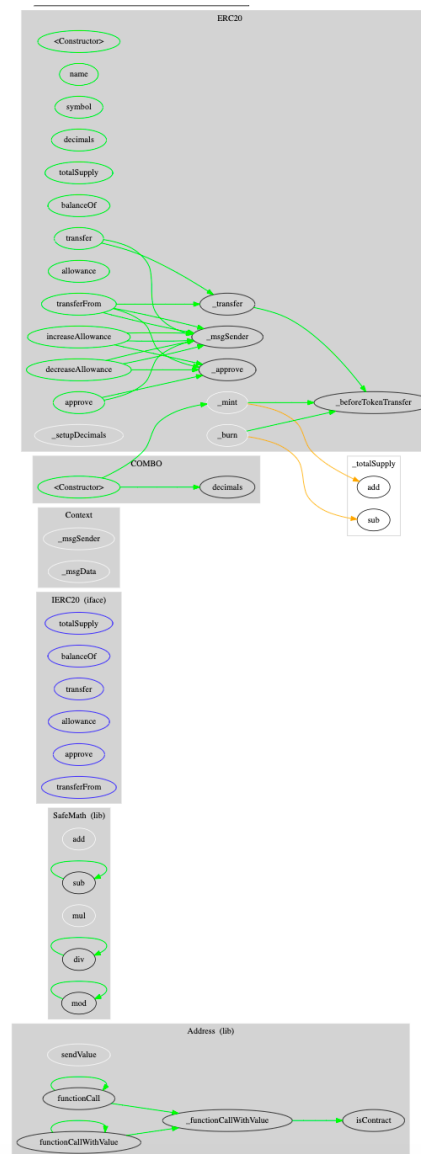
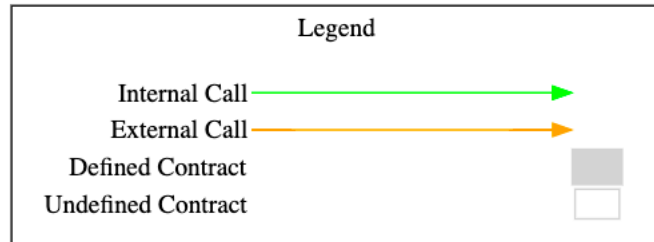
Dependency / Import Path	Source
@openzeppelin/contracts/utils/Address.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v3.2.0/contracts/utils/Address.sol
@openzeppelin/contracts/math/SafeMath.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v3.2.0/contracts/math/SafeMath.sol
@openzeppelin/contracts/token/ERC20/IERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v3.2.0/contracts/token/ERC20/IERC20.sol
@openzeppelin/contracts/GSN/Context.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v3.2.0/contracts/GSN/Context.sol
@openzeppelin/contracts/token/ERC20/ERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v3.2.0/contracts/token/ERC20/ERC20.sol

4.3 Tested Contract Files

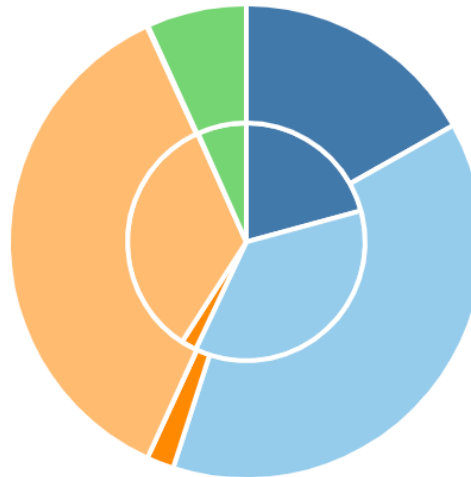
The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability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

File	Fingerprint (MD5)
COMBO.sol	929037e17edacfd41743cc108fc3b604











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
<div><code>^0.6.2</code></div> <div><code>^0.6.0</code></div>				<div><code>yes</code></div> <div>(2 asm blocks)</div>	
 Transfers ETH	 Low-Level Calls	 DelegateCall	 Uses Hash Functions	 ECTrecover	 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				
19	0				
External	Internal	Private	Pure	View	
6	44	1	8	12	

Components

 Contracts	 Libraries	 Interfaces	 Abstract
2	2	1	1

4.7 Metrics / Source Unites in Scope

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
	COMBO.sol	5	1	736	670	219	455	151	
	Totals	5	1	736	670	219	455	151	

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 Furucombo Team provided us with the files that needs to be tested. The scope of the audit is the COMBO Token contract.

Following contracts with the direct imports has been tested:

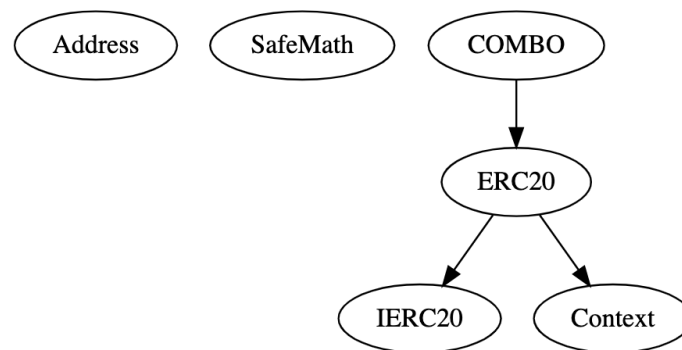
- COMBO.sol

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

Verify claims:

1. ERC-20 Token standard is correct implemented
2. Deployer cannot mint any new tokens.
3. Deployer cannot burn or lock user funds
4. Deployer cannot pause the contract
5. Overall smart contract security needs to be checked

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





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

5.2. SWC Attacks & Special Checks

ID	Title	Relationships	Test Result
SWC-131	Presence of unused variables	CWE-1164: Irrelevant Code	
SWC-130	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	
SWC-129	Typographical Error	CWE-480: Use of Incorrect Operator	
SWC-128	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	
SWC-127	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	
SWC-125	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	
SWC-124	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	
SWC-123	Requirement Violation	CWE-573: Improper Following of Specification by Caller	


ID	Title	Relationships	Test Result
SWC-122	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	✓
SWC-121	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	✓
SWC-120	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	✓
SWC-119	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards	✓
SWC-118	Incorrect Constructor Name	CWE-665: Improper Initialization	✓
SWC-117	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature	✓
SWC-116	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	✓
SWC-115	Authorization through tx.origin	CWE-477: Use of Obsolete Function	✓
SWC-114	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	✓

ID	Title	Relationships	Test Result
SWC-113	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	✓
SWC-112	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	✓
SWC-111	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	✓
SWC-110	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	✓
SWC-109	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	✓
SWC-108	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	✓
SWC-107	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	✓
SWC-106	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	✓
SWC-105	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	✓
SWC-104	Unchecked Call Return Value	CWE-252: Unchecked Return Value	✓


ID	Title	Relationships	Test Result
SWC-103	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	
SWC-102	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	
SWC-101	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	
SWC-100	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	

6. Verify Claims

6.1 ERC-20 Token standard is correct implemented

Status: tested and verified 

6.2 Deployer cannot mint any new tokens.

Status: tested and verified 

Max / Initial Supply: 100000000 (100M)

Code: Ln 727 – 736

```
contract COMBO is ERC20 {
    using Address for address;
    using SafeMath for uint256;


    constructor() public ERC20("Furucombo", "COMBO") {
        // mint amount 100M
        uint256 supply = 100000000 * (10**uint256(decimals()));
        _mint(msg.sender, supply);
    }
}

function _mint(address account, uint256 amount) internal virtual {
    require(account != address(0), "ERC20: mint to the zero address");

    _beforeTokenTransfer(address(0), account, amount);

    _totalSupply = _totalSupply.add(amount);
    _balances[account] = _balances[account].add(amount);
    emit Transfer(address(0), account, amount);
}
```

6.3 Deployer cannot burn or lock user funds

Status: tested and verified 

Code: No burn or lock function

1. approve →

2. decreaseAllowance →


3. increaseAllowance →

4. transfer →

5. transferFrom →

Powered by [Etherscan.io](https://etherscan.io). Browse [source code](#)

6.4 Deployer cannot pause the contract

Status: tested and verified 

Code: No pause function

1. approve →

2. decreaseAllowance →


3. increaseAllowance →

4. transfer →

5. transferFrom →

Powered by [Etherscan.io](https://etherscan.io). Browse [source code](#)

6.5 Overall smart contract security needs to be checked

Status: tested and verified 



7. Executive Summary

Two (2) independent Chainsulting experts performed an unbiased and isolated audit of the smart contract codebase. The final debrief took place on the March 27, 2021. The overall code quality of the project is very 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 and for safe mathematical operations.

The main goal of the audit was to verify the claims regarding the security of the smart contract and the claims inside the scope of work. During the audit, no issues were found after the manual and automated security testing.

8. Deployed Smart Contract

VERIFIED

Smart Contract is deployed here:

<https://etherscan.io/address/0xffffffff2ba8f66d4e51811c5190992176930278#code>

