LaunchX (LNCHX)

SMART CONTRACT AUDIT

11.05.2021

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 LaunchX Finance. 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 (11.05.2021)	Layout
0.5 (11.05.2021)	Verify Claims and Test Deployment
0.6 (11.05.2021)	Testing SWC Checks
0.8 (11.05.2021)	Automated Security Testing Manual Security Testing
0.9 (11.05.2021)	Summary and Recommendation
1.0 (11.05.2021)	Final document

2. About the Project and Company

Company address:

LaunchX Limited
Fortgate Offshore Investment and Legal Services Ltd
1st Floor, The Sotheby Building
Rodney Bay, Gros Islet
P.O. BOX 838
CASTRIES SAINT LUCIA

Website: https://launchx.finance

Twitter: https://twitter.com/LaunchX_

Medium: https://launchx.medium.com

Telegram: https://t.me/weLaunchX

2.1 Project Overview

LaunchX is an all-in-one cryptocurrency service that focuses on token issuance, cross-chain synthetic staking & interoperability, and a governance model that allows for a transparent and fair listing process. LaunchX is ran by its community through a first-of-its-kind DAO protocol. It’s also known as Multi-Chain Decentralized IDO Launchpad Platform for Ethereum, Binance Smart Chain, Solana and Polkadot.

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)

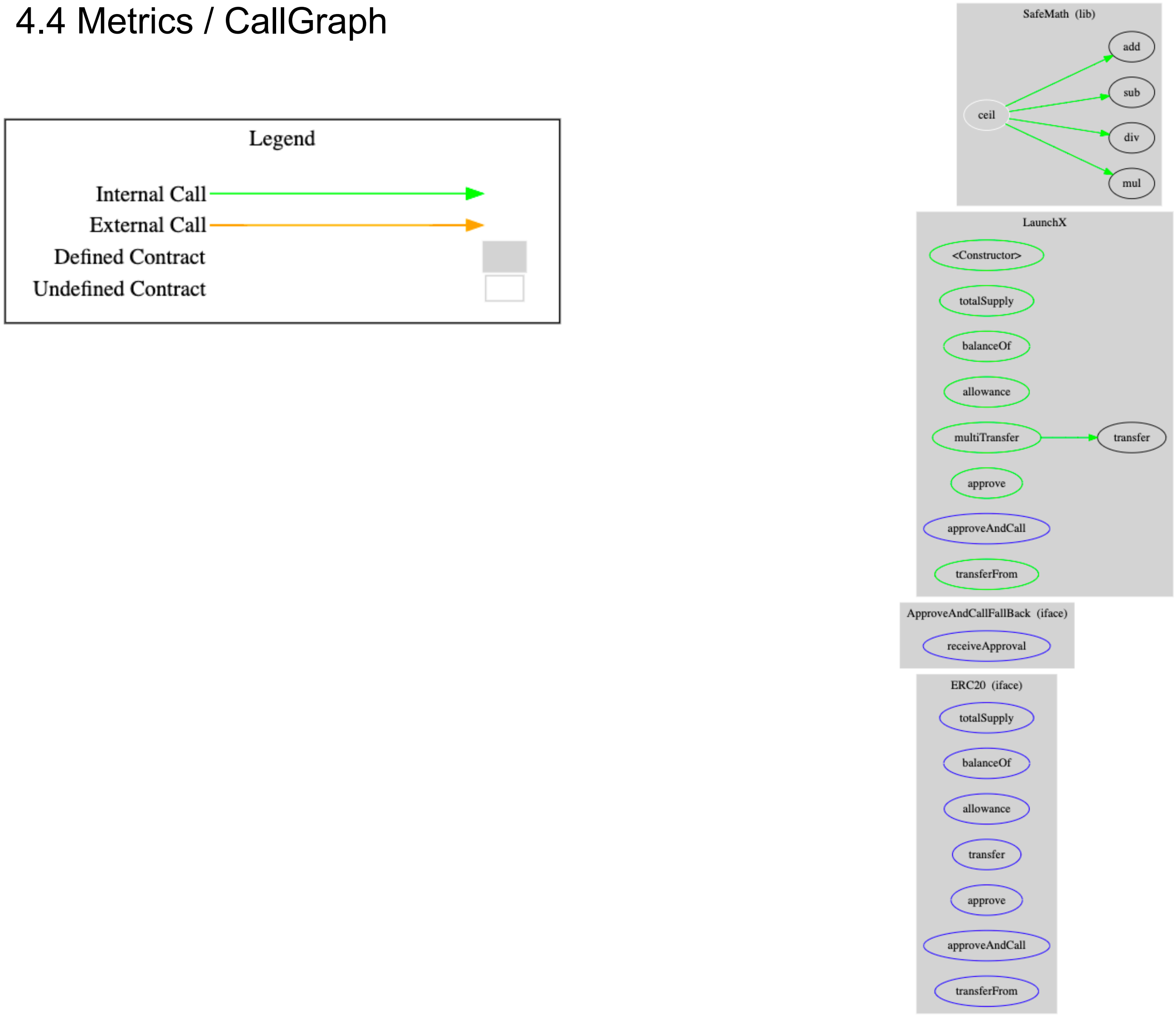
Dependency / Import Path	Source
@openzeppelin/contracts/token/ERC20/IERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v2.0/contracts/token/ERC20/IERC20.sol
@openzeppelin/contracts/token/ERC20/ERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v2.0/contracts/token/ERC20/ERC20.sol
@openzeppelin/contracts/math/SafeMath.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v2.0.0/contracts/math/SafeMath.sol

4.3 Tested Contract Files

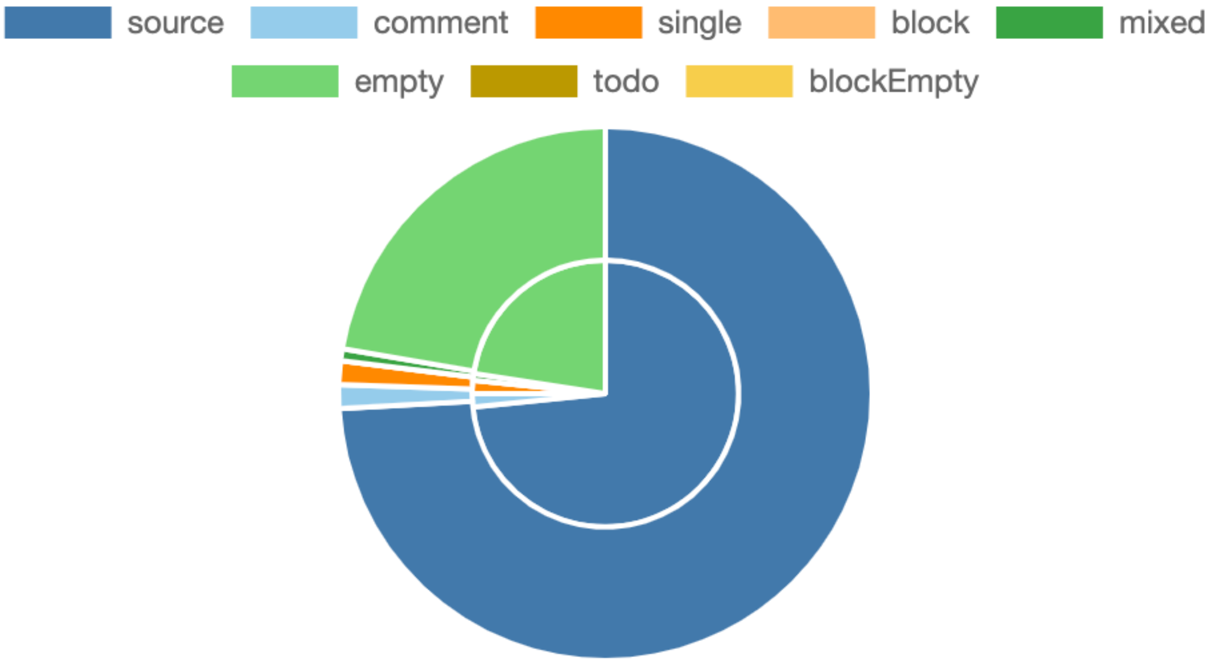
The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability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

File	Fingerprint (MD5)
LNCHX.sol	2a1a9da4c8db2833af060568a5f767ca

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
`^0.4.25`			**** (0 asm blocks)	

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

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
17	0

External	Internal	Private	Pure	View
9	16	0	5	6

StateVariables

Total	🌐Public
7	3

4.7 Metrics / Source Unites in Scope

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📝📚🔍	LaunchX Token/LNCHX.sol	2	2	135	120	91	2	83	📤
📝📚🔍	**Totals**	**2**	**2**	**135**	**120**	**91**	**2**	**83**	📤

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 LaunchX Team provided us with the file that needs to be tested. The scope of the audit is the LNCHX Token contract.

Following contracts with the direct imports has been tested:

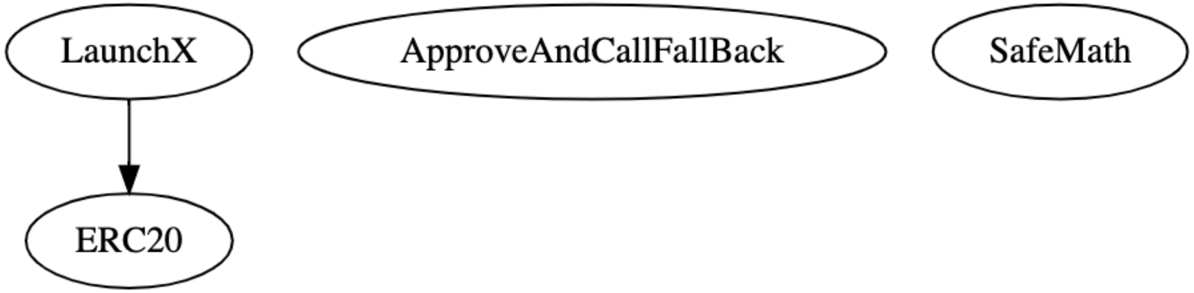
- LNCHX.sol

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

Verify claims:

1. BEP-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

5.1.1 Wrong import of OpenZeppelin library

Severity: Informational
Status: Acknowledged
File(s) affected: All

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation, OpenZeppelin files are added directly into the code. This violates OpenZeppelin's MIT license, which requires the license and copyright notice to be included if its code is used. Moreover, updating code manually is error-prone.	`ERC20, SafeMath, IERC20`	We recommend using npm (import "@openzeppelin/contracts/..) in order to guarantee that original OpenZeppelin contracts are used with no modifications. This also allows for any bug-fixes to be easily integrated into the codebase.

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	X
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 BEP-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: 75,000,000 (75M)
Code: Ln 33 / 35 - 38

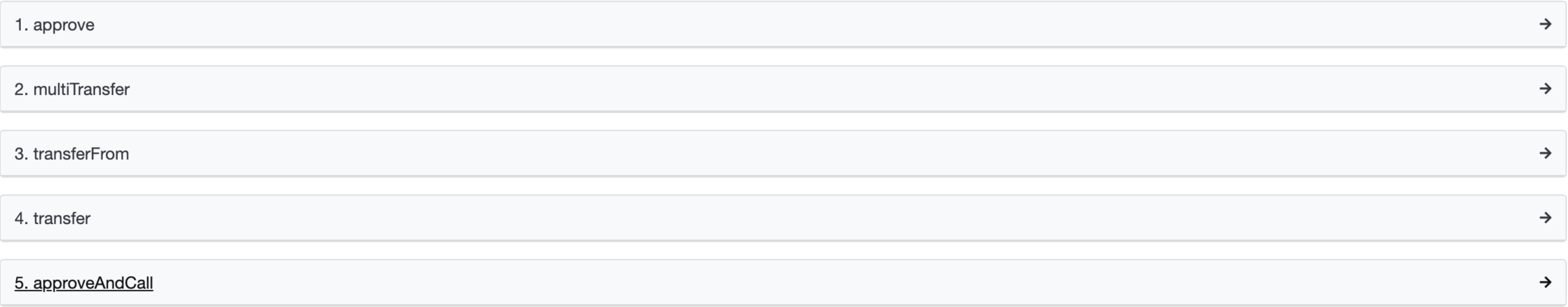
```
uint256 _totalSupply = 75000000 * (10**18); // 75 million LAUNCHX token supply

constructor() public {
    balances[msg.sender] = _totalSupply;
    emit Transfer(address(0), msg.sender, _totalSupply);
  }
```

6.3 Deployer cannot burn or lock user funds

Status: tested and verified ✅

Code: No burn or lock function



1. approve →

2. multiTransfer →

3. transferFrom →

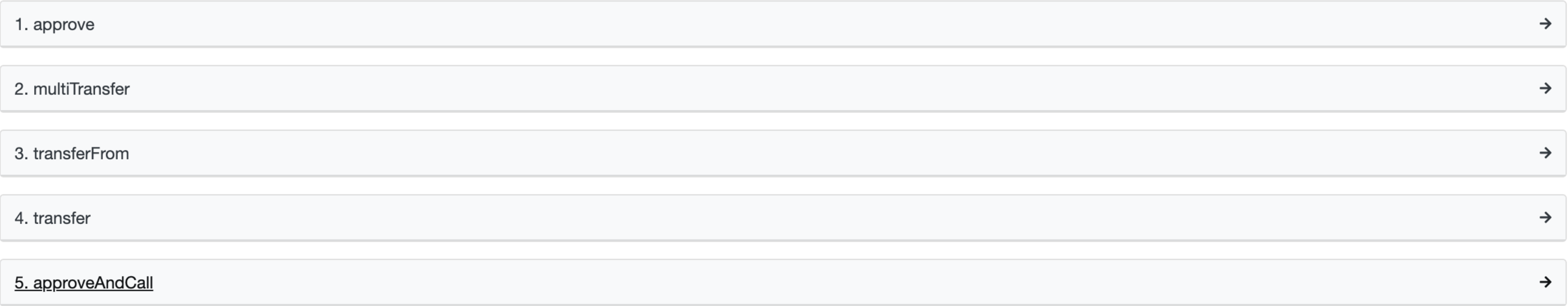
4. transfer →

5. approveAndCall →

6.4 Deployer cannot pause the contract

Status: tested and verified ✅

Code: No pause function



1. approve →

2. multiTransfer →

3. transferFrom →

4. transfer →

5. approveAndCall →

6.5 Overall smart contract security needs to be checked

Status: tested and verified ✅

7. Executive Summary

Chainsulting experts performed an unbiased and isolated audit of the smart contract codebase. The final debriefs took place on the May 11, 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 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://bscscan.com/address/0xC43570263e924C8cF721F4b9c72eEd1AEC4Eb7DF#code