Blaize.Security

April 9th, 2022 / V. 1.0

CRYPTO COLLECTIVE SMART CONTRACT AUDIT

TABLE OF CONTENTS

AUDIT RATING

CryptoCollective contract's source code was taken from the repository provided by the CryptoCollective team.

SCORE **9.9** /10

The scope of the project is **CryptoCollective** set of contracts**:**

1/ CryptoCollectiveNFT

Contracts were delivered from the repository with contract and tests.
https://github.com/cryptocollective/contract

Initial commit:

- df55f90090949d1ece16afc96b4a42586241de0f

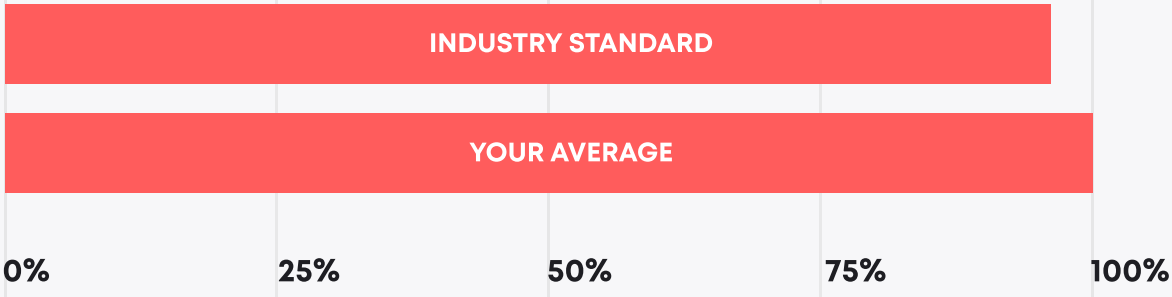
Last reviewed commit:

- 7e70be7687943eee9cef438bd0049f0c66cbb9a0

TECHNICAL SUMMARY

In this report, we consider the security of the contracts for CryptoCollective protocol. Our task is to find and describe security issues in the smart contracts of the platform. This report presents the findings of the security audit of **CryptoCollective** smart contracts conducted between **April 8th, 2022 - April 9th, 2022.**

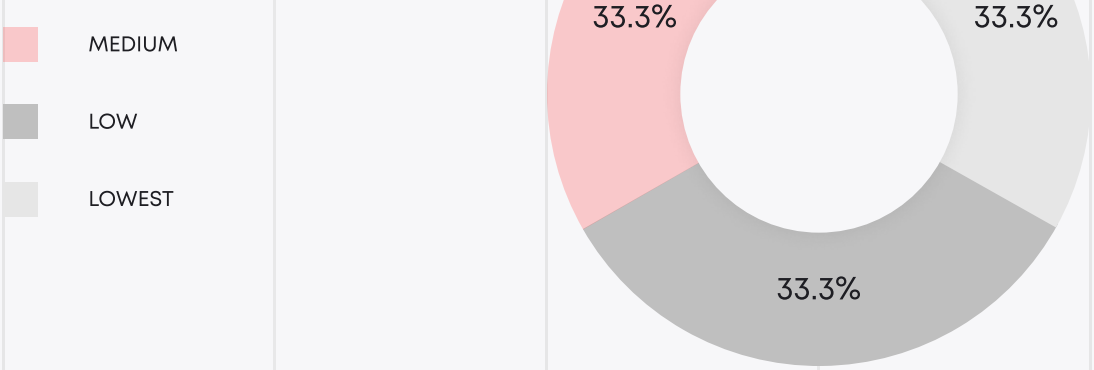
Testable code



The testable code is 100%, which is above the industry standard of 95%.

The scope of the audit includes the unit test coverage, that bases on the smart contracts code, documentation and requirements presented by the CryptoCollective team. Coverage is calculated based on the set of Truffle framework tests and scripts from additional testing strategies. Though, in order to ensure a security of the contract Blaize.Security team recommends the CryptoCollective team put in place a bug bounty program to encourage further and active analysis of the smart contracts.

THE GRAPH OF VULNERABILITIES DISTRIBUTION:



The table below shows the number of found issues and their severity. A total of 3 problems were found. 3 issues were fixed or verified by the CryptoCollective team.

	FOUND	FIXED/VERIFIED
Critical	0	0
High	0	0
Medium	1	1
Low	1	1
Lowest	1	1

SEVERITY DEFINITION

Critical

A system contains several issues ranked as very serious and dangerous for users and the secure work of the system. Needs immediate improvements and further checking.

High

A system contains a couple of serious issues, which lead to unreliable work of the system and migh cause a huge information or financial leak. Needs immediate improvements and further checking.

Medium

A system contains issues which may lead to mediumfinancial loss or users' private information leak. Needs immediate improvements and further checking.

Low

A system contains several risks ranked as relatively small with the low impact on the users' information and financial security. Needs improvements.

Lowest

A system does not contain any issue critical to the secure work of the system, yet is relevant for best

AUDITING STRATEGYAND TECHNIQUES APPLIED \ PROCEDURE

We have scanned this smart contract for commonly known and more specific vulnerabilities:

- Unsafe type inference;
- Timestamp Dependence;
- Reentrancy;
- Implicit visibility level;
- Gas Limit and Loops;
- Transaction-Ordering Dependence;
- Unchecked external call - Unchecked math;
- DoS with Block Gas Limit;
- DoS with (unexpected) Throw;
- Byte array vulnerabilities;
- Malicious libraries;
- Style guide violation;
- ERC20 API violation;
- Uninitialized state/storage/ local variables;
- Compile version not fixed.

Procedure

In our report we checked the contract with the following parameters:

- Whether the contract is secure;
- Whether the contract corresponds to the documentation;
- Whether the contract meets best practices in efficient use of gas, code readability;

Automated analysis:

Scanning contract by several public available automated analysis tools such as Mythril, Solhint, Slither and Smartdec. Manual verification of all the issues found with tools.

Manual audit:

Manual analysis of smart contracts for security vulnerabilities. Checking smart contract logic and comparing it with the one described in the documentation.

EXECUTIVE SUMMARY

The contract represents upgradeable NFT (corresponding to ERC721 and ERC1155 standards) with custom minting mechanism, which relies on several rounds and verified signatures. The contract has good code quality and has full native unit-test coverage.

All issues found during the audit were connected to the reentrancy probability, which was marked as medium and low risk due to specificalty of the conditions to recreate the exploit.
Nevertheless, issues were either resolved or verified by the CryptoCollective team. Style issue left has no impact on the contracts security.

	RATING
Security	9.9
Gas usage and logic optimization	9.7
Code quality	10
Test coverage**	10
Total	9.9

** Contracts have good native coverage which was checked within the scope of the audit. Nevertheless - security team has prepared own set of tests.

COMPLETE ANALYSIS

MEDIUM	✔ Verified

Return extra ETH to the message sender.

CryptoCollectiveNFT.sol: function claim().
In case message sender has sent more ETH than required(assertion in Line 123) exceeded ETH should be sent back to the message sender at the end of function execution(In order to avoid reentrancy vulnerability).

Post-audit:

Extra ETH verified to be withdrawn in the separate withdrawETH() function by the owner.

LOW	✔ Resolved

Possible Reentrancy attack.

CryptoCollectiveNFT.sol: function claim().
Contract CryptoCollectiveNFT.sol inherits ERC1155Upgradeable.sol from OpenZeppelin library. In ERC1155Upgradeable.sol there is an external call to an address which receives tokens in function _doSafeTransferAcceptanceCheck() (Which is called within function _mint()). Round information for the message caller about purchased tokens is updated after _mint() is called, which can lead to possible reentrancy attacks. Issue is marked as low, since only users with a message, signed by the signer, are able to call function claim() and purchase tokens.

Recommendation:

In order to protect the contract itself, either update info in mapping “rounds” before minting is performed or use NonReentrant modifier from OpenZeppelin contract ReentrancyGuardUpgradeable.sol. It is also important to verify addresses, which receive signed messages from the signer, so that those addresses are not malicious contracts.

LOWEST	✔ Verified

Enum can be used.

CryptoCollectiveNFT.sol: Lines 23-25.
Instead of defining each constant separately, enum can be used as a set of constants for such values which starts from 0 and increments by 1.

Recommendation:

Use enum instead of defining constants.

Post-audit:

After the conversation with the CryptoCollective team, constants usage was justified.

	CryptoCollectiveNFT
✔ Re-entrancy	Pass
✔ Access Management Hierarchy	Pass
✔ Arithmetic Over/Under Flows	Pass
✔ Delegatecall Unexpected Ether	Pass
✔ Default Public Visibility	Pass
✔ Hidden Malicious Code	Pass
✔ Entropy Illusion (Lack of Randomness)	Pass
✔ External Contract Referencing	Pass
✔ Short Address/ Parameter Attack	Pass
✔ Unchecked CALL Return Values	Pass
✔ Race Conditions / Front Running	Pass
✔ General Denial Of Service (DOS)	Pass
✔ Uninitialized Storage Pointers	Pass
✔ Floating Points and Precision	Pass
✔ Tx.Origin Authentication	Pass
✔ Signatures Replay	Pass
✔ Pool Asset Security (backdoors in the underlying ERC-20)	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

As part of the audit process, Auditors team has checked and verified existing native unit-test coverage. It was verified to sufficient for the security purpose, contains all necessary tests to cover the business logic of the contract,

Contract: CryptoCollectiveNFT

Native coverage

- ✓ has maxSupply set after initialize (448ms)
- ✓ tokenURI of inner (1113ms)
- ✓ tokenURI of collective (945ms)
- ✓ name (352ms)
- ✓ symbol (362ms)
- ✓ opensea (382ms)
- ✓ round (403ms)
- ✓ signer (410ms)
- ✓ can claim (710ms)
- ✓ can claim with desired (753ms)
- ✓ can not claim with incorrect signer (399ms)
- ✓ claim requires correct round (331ms)
- ✓ sets round data correctly after claim (496ms)
- ✓ can not claim when paused (863ms)
- ✓ can change min payable (767ms)
- ✓ denies claiming more then allowed (542ms)
- ✓ denies claiming more then allowed via split (827ms)
- ✓ can claim under max supply (3739ms)
- ✓ has a balance when paid (835ms)
- ✓ can claim more in new round (2497ms)

20 passing (22s)

Auditors team has provided extra testing to check, that It is not possible to buy more NFTs than allowed by signer in one round (especially with the same signed message).

Additional test-cases

- ✓ Cannot claim more than allowed in one round (1229ms)
- ✓ Round data updates correctly (1156ms)

2 passing(4s)

TEST COVERAGE RESULTS

FILE	% STMTS	% BRANCH	% FUNCS
CryptoCollectiveNFT.sol	100	95.45	100
All files	**100**	**95.45**	**100**

The result includes CryptoCollective own set of unit tests, additional tests by Blaize Security.

DISCLAIMER

The information presented in this report is an intellectual property of the customer including all presented documentation, code databases, labels, titles, ways of usage as well as the information about potential vulnerabilities and methods of their exploitation. This audit report does not give any warranties on the absolute security of the code. Blaize.Security is not responsible for how you use this product and does not constitute any investment advice.

Blaize.Security does not provide any warranty that the working product will be compatible with any software, system, protocol or service and operate without interruption. We do not claim the investigated product is able to meet your or anyone else requirements and be fully secure, complete, accurate and free of any errors and code inconsistency.

We are not responsible for all subsequent changes, deletions and relocations of the code within the contracts that are the subjects of this report.

You should perceive Blaize.Security as a tool which helps to investigate and detect the weaknesses and vulnerable parts that may accelerate the technology improvements and faster error elimination.