DIA DATA

DRM NFT

SMART CONTRACT AUDIT

04.09.2021

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 D.I.A. e.V. . 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 (04.06.2021)	Layout
0.4 (04.06.2021)	Automated Security Testing Manual Security Testing
0.5 (08.06.2021)	Verify Claims and Test Deployment
0.6 (08.06.2021)	Testing SWC Checks
0.9 (09.06.2021)	Summary and Recommendation
1.1 (09.06.2021)	Final document
2.0 (04.09.2021)	Re-check
2.1 (TBA)	Added deployed contract

2. About the Project and Company

Company address:

D.I.A. e.V. (Association)
Baarerstrasse 10
6300 Zug
Switzerland

Website: https://diadata.org

Twitter: https://twitter.com/diadata_org

Medium: https://medium.com/@diadata_org

Telegram: https://t.me/DIAdata_org

LinkedIn: https://www.linkedin.com/company/diadata-org

GitHub: https://github.com/diadata-org/diadata

Reddit: https://www.reddit.com/user/DIAdata

YouTube: https://www.youtube.com/c/DIAdata_org

2.1 Project Overview

DIA (Decentralised Information Asset) is an open-source oracle platform that enables market actors to source, supply and share trustable data. DIA aims to be an ecosystem for open financial data in a financial smart contract ecosystem, to bring together data analysts, data providers and data users. In general, DIA provides a reliable and verifiable bridge between off-chain data from various sources and on-chain smart contracts that can be used to build a variety of financial DApps. DIA is the governance token of the platform. It is currently based on ERC-20 Ethereum protocol. The project was founded in 2018, while the token supply was made available to the public during the bonding curve sale from Aug. 3 through Aug. 17, 2020, where 10.2 million tokens were sold.

Who Are the Founders of DIA?
The DIA association was co-founded by a group of a dozen people, though Paul Claudius, Michael Weber and Samuel Brack are the leaders. Claudius is the face of the project and its lead advocate, sometimes also mentioned as a CBO. He has a masters degree in international management from ESCP Europe and a bachelors in business and economics from Passau University. Apart from working on DIA, he is also a co-founder and CEO of BlockState AG and c ventures. Before crypto, he had worked as director for a nutrition company called nu3. Weber is the project's CEO. He holds a asters in management from ESCP Business School and an equivalent to a bachelors in economics and physics from University of Cologne. He has worked in several banks and financial institutions before turning to crypto, where he founded such projects as Goodcoin, myLucy and BlockState. Samuel Brack serves DIA in the role of CTO. Like both Claudius and Weber, he shares the same position at BlockState. He has a masters degree in computer science from Humboldt University of Berlin, where as of January 2020, he is still studying for his PhD.

What Makes DIA Unique?
DIA aims to become the Wikipedia of financial data. It specifically addresses the problem of dated/unverified/hard to access data in the world of finance and crypto, especially DeFi, while proposing to solve it via system of financial incentives for users to keep the flow of open-source, validated data streams to the oracles up and running. The current design of oracles, DIA argues, is non-transparent, difficult to scale and vulnerable to attack. The DIA governance token will be used to fund data collection, data validation, voting on governance decisions and to incentivize the development of the platform. Users can stake DIA tokens to incentivise new data to appear on the platform, but access to historical data though DIA is free.

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

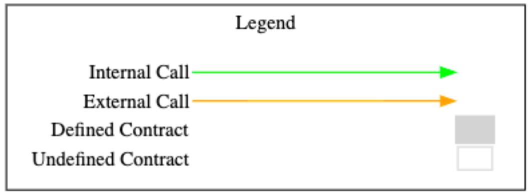
Dependency / Import Path	Source
@openzeppelin/contracts/access/Ownable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable.sol
@openzeppelin/contracts/math/Math.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/math/Math.sol
@openzeppelin/contracts/token/ERC20/ERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/ERC20.sol
@openzeppelin/contracts/token/ERC1155/ERC1155.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC1155/ERC1155.sol

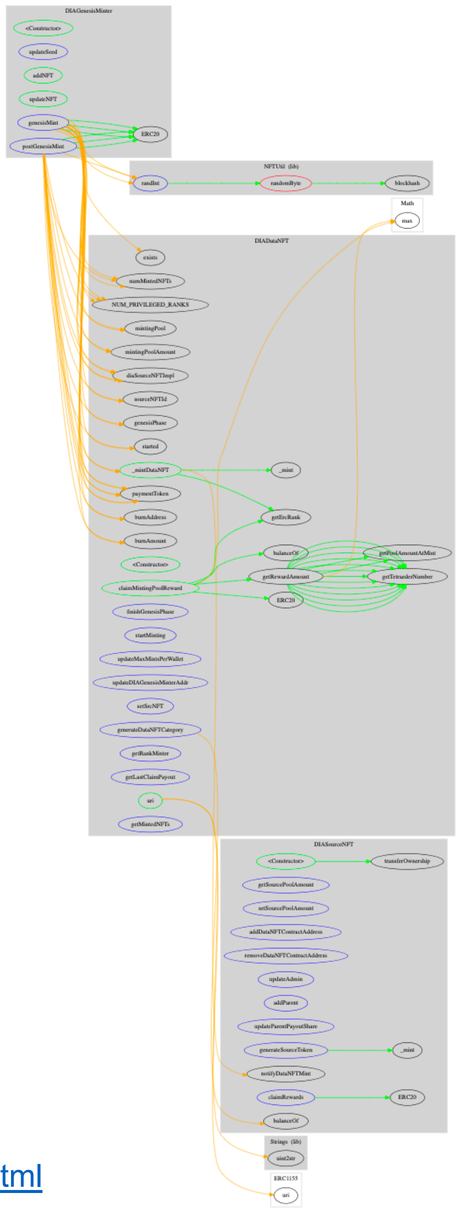
4.3 Tested Contract Files

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)
./DIADataNFT.sol	e84371bf242e82f7dc9fce6870cc9423
./DIASourceNFT.sol	8c71c936aebffa69281dea2709a5d2c1
./DIAGenesisMinter.sol	f87bf1c75d1aeee0c970209f162ffe9f
./Strings.sol	a2e804893783a443b836acf1ddd361a8

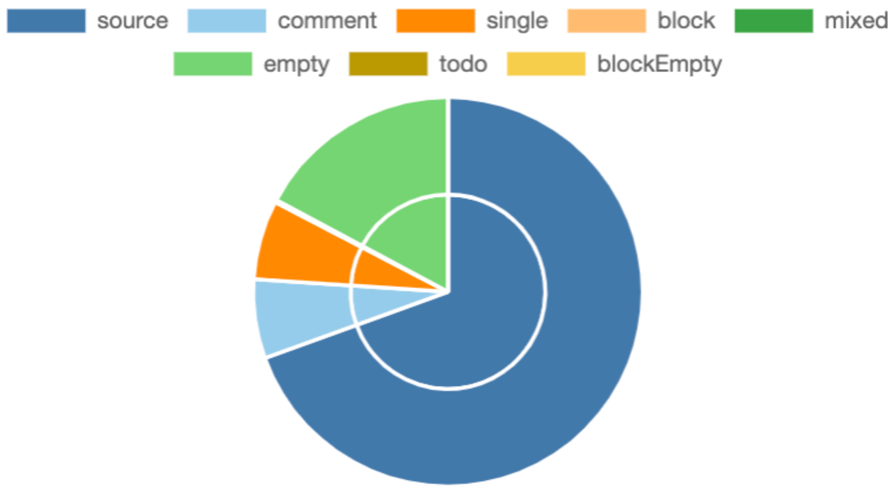
4.4 Metrics / CallGraph

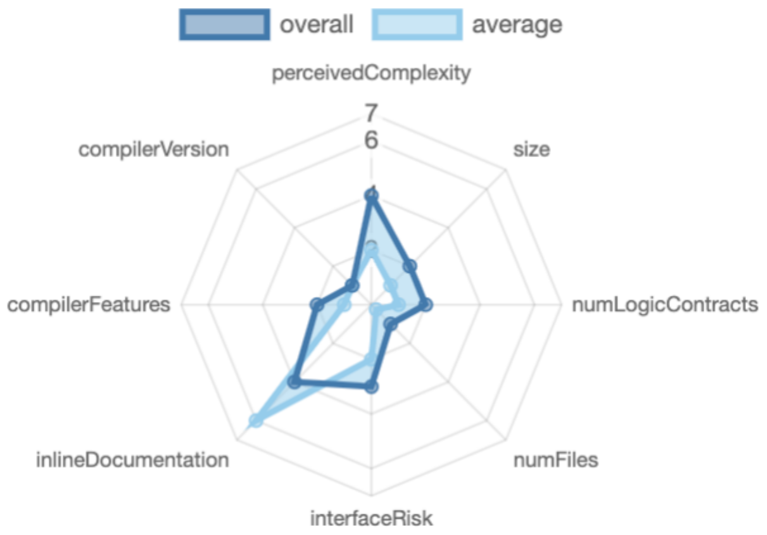




View full version: https://chainsulting.de/wp-content/uploads/2021/09/solidity-metrics_dia.html

4.5 Metrics / Source Lines & Risk





4.6 Metrics / Capabilities

Solidity Versions observed	🧪 Experimental Features	💰 Can Receive Funds	🖥 Uses Assembly	💣 Has Destroyable Contracts
`0.8.0`			**** (0 asm blocks)	

📤 Transfers ETH	⚡ Low-Level Calls	👥 DelegateCall	🧮 Uses Hash Functions	🔖 ECRecover	🌀 New/Create/Create2
`yes`			`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
30	0

External	Internal	Private	Pure	View
23	28	1	3	9

StateVariables

Total	🌐Public
25	21

4.7 Metrics / Source Unites in Scope

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📝📚	contracts/DIAGenesisMinter.sol	2		113	113	79	12	113	🧮
📚	contracts/Strings.sol	1		28	28	24	1	18	
📝	contracts/DIASourceNFT.sol	1		117	117	92	4	66	📤
📝	contracts/DIADataNFT.sol	1		208	208	153	16	128	
📝📚	**Totals**	**5**		**466**	**466**	**348**	**33**	**325**	📤🧮

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, ...)

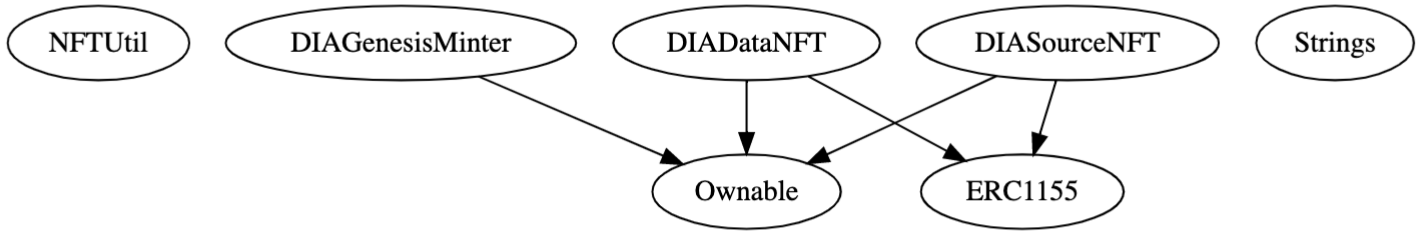
5. Scope of Work

The DIA Data Team provided us with the file that needs to be tested. The scope of the audit are the DRM NFT contracts.

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

- Source NFTs are held by data owners and are used to control the licensing of data to DIA API data users
- Only the owner of the Source NFTs should be able to create new categories
- Only the contract owner can generate new source NFTs
- Source NFTs can have parent source NFTs. In case a source NFT has multiple parents, its source rewards are split evenly between all parents
- Data NFTs are minted by data users. They pay for the minting and participate by claiming future rewards from the minting pool
- 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

5.1.1 Weak PRNG

Severity: High
File(s) affected: DIADataNFT.sol
Status: FIXED (The randomness is not important for the use-case)

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation, there is a weak source of randomness due to the use of block.timestamp and blockhash. These parameters can be influenced by miners to some extent so they should be avoided.	Line 13: `return uint8(uint256(keccak256(abi.encodePacked(block.timestamp, blockhash(block.number - 1), msg.sender, seed)))%256);`	We highly recommend not to use block.timestamp, now or blockhash as a source of randomness.

5.1.2 Unchecked token transfer

Severity: High
File(s) affected: DIADataNFT.sol, DIASourceNFT.sol
Status: FIXED

Attack / Description	Code Snippet	Result/Recommendation

In the current implementation, the return value of the external transferFrom function call is not checked. Several tokens do not revert in case of failure and return false.	DIADataNFT.sol Line 118:
	`ERC20(paymentToken).transferFrom(msg.sender, burnAddress, burnAmount);`
	
	DIADataNFT.sol Line 120:
	`ERC20(paymentToken).transferFrom(msg.sender, mintingPool, mintingPoolAmount);`
	
	DIADataNFT.sol Line 133:
	`ERC20(paymentToken).transferFrom(msg.sender, diaSourceNFTImpl.sourcePool(), diaSourceNFTImpl.getSourcePoolAmount(currSourceNFTId));`
	
	DiaSourceNFT.sol Line 108:
	`ERC20(paymentToken).transferFrom(sourcePool, claimer, payoutDataTokens);`
		We highly recommend to use OpenZeppelins *SafeERC20*, or ensure that the transfer/transferFrom return value is checked.

MEDIUM ISSUES

5.1.3 Division before multiplication

Severity: Medium
File(s) affected: DIADataNFT.sol
Status: FIXED

Attack / Description	Code Snippet	Result/Recommendation
Solidity integer division might truncate. As a result, performing multiplication before division can sometimes avoid loss of precision.	Line 173: `uint rawClaim = (ownRawWeight / sumAllWeights) * getPoolAmountAtMint(maxNumMinted);` Line 181: `uint rawClaimLastClaim = (ownRawWeightLastClaim / sumAllWeightsLastClaim) * getPoolAmountAtMint(numMintedLastClaim);`	We highly recommend ordering multiplications before division.

5.1.4 Unsecure arithmetic operations

Severity: Medium
File(s) affected: DIADataNFT.sol, DIASource.sol
Status: FIXED (Solidity 0.8.0 has integrated SafeMath)

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation arithmetic operations are unsecure due to potential under- or overflow.	overall	We recommend using OpenZeppelins library SafeMath for all arithmetic operation to ensure safe calculations.

LOW ISSUES

5.1.5 Missing zero-address check

Severity: LOW

File(s) affected: DIADataNFT.sol, DIASourceNFT.sol

Status: FIXED

Attack / Description	Code Snippet	Result/Recommendation

In the current implementation, there are several addresses set without checking for the zero address. This can lead to unintended behaviour.	DIADataNFT.sol Line 51 & 52 & 55:
	`paymentToken = _paymentToken;`
	`burnAddress = _burnAddress;`
	`mintingPool = _mintingPool;`
	
	DIADataNFT.sol Line 74:
	`burnAddress = newBurnAddress;`
	
	DIASourceNFT.sol Line 31:
	`dataNFTContractAddress = _dataNFTContractAdd`
	`ress;`
	
	DIASourceNFT.sol Line 39:
	`dataNFTContractAddress = newAddress;`
		We recommend checking addresses for the zero address with require statements before setting them as a variable.

5.1.6 Hardcoded URI
Severity: LOW
File(s) affected: DIASourceNFT.sol
Status: ACKNOWLEDGED

Attack / Description	Code Snippet	Result/Recommendation

In the current implementation, there is a hardcoded URI which gives you less flexibility in the future or the domain api.diadata.org can be hijacked.	Line 26 - 28
	`constructor(address newOwner) ERC1155("https://api.diadata.org/v1/nft/source_{id}.json") {`
	`        transferOwnership(newOwner);`
	`    }`
		We recommend to put the URL part (`api.diadata.org/v1/nft/source_{`) into a string as well, so the Owner is able to adjust it, in case of emergency or an upgrade.

INFORMATIONAL ISSUES

5.1.7 Violated naming conventions
Severity: Informational
File(s) affected: DIADataNFT.sol, DIASourceNFT.sol
Status: FIXED

Attack / Description	Code Snippet	Result/Recommendation

Solidity defines a naming convention that should be followed. In the current implementation structs are written in mixedCase.	DIADataNFT.sol Line 32:
	`struct dataNFTCategory {`
	
	DIASourceNFT.sol Line 15:
	`struct sourceNftMetadata {`
		We recommend following the solidity naming conventions for better code readability. Structs should be written in CapWords such as *DataNFTCategory*. More information about naming conventions on https://docs.soliditylang.org/en/v0.4.25/style-guide.html#naming-conventions.

5.1.8 Public functions could be external

Severity: Informational
File(s) affected: DIADataNFT.sol, DIASourceNFT.sol
Status: FIXED

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation several functions are declared as public where they could be external.	DIADataNFT.sol Lines: 16, 65, 69, 73, 77, 86, 91, 138, 195 DIASourceNFT.sol Lines: 34, 38, 42, 47, 57, 84, 92, 102	We recommend declaring functions as external for better code readability.

5.1.9 Missing natspec documentation

Severity: Informational
File(s) affected: DIADataNFT.sol, DIASourceNFT.sol, Strings.sol, DIAGenesisMinter.sol
Status: FIXED

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.

5.1.10 Unexplicit variable typs

Severity: Informational
File(s) affected: DIADataNFT.sol
Status: FIXED

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation all integer variables have an unexplicit type (uint).	Overall	It is recommended to use explicit variable declaration such as uint8 or uint256. It improves code readability and can help to make sure variables have a intended size.

5.1.11 Variable burn address

Severity: Informational
File(s) affected: DIADataNFT.sol
Status: FIXED

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation the *burnAddress* variable is set to an address passed to the constructor. It makes it possible to pass other addresses instead of the zero address by contract creation to claim tokens, which should be burned.	Line 52: `burnAddress = _burnAddress;`	It is recommended to set the *burnAddress* to the zero address hardcoded in the contract. This leads to a guaranteed burning of the tokens by sending them directly to the zero address.

5.1.12 Unused variable

Severity: Informational
File(s) affected: DIADataNFT.sol
Status: FIXED

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation the *sourcePoolAmount* variable is set in the constructor but never used again in the contract.	Line 56: `sourcePoolAmount = _sourcePoolAmount;`	It is recommended to remove all unused variables.

5.1.13 Unnecessary alias variable

Severity: Informational
File(s) affected: DIASourceNFT.sol
Status: FIXED

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation the *sourcePool* variable is set to address(this) in the constructor and in the contract only used once.	Line 30: `sourcePool = address(this);`	It is recommended to remove the alias variable *sourcePool* and just use address(this). Removing variables saves gas by contract creation.

5.2 Verify claims

5.2.1 Source NFTs are held by data owners and are used to control the licensing of data to DIA API data users
Status: tested and verified ✅

5.2.2 Only the owner of the Source NFTs should be able to create new categories
Status: tested and verified ✅

5.2.3 Only the contract owner can generate new source NFTs
Status: tested and verified ✅

5.2.4 Source NFTs can have parent source NFTs. In case a source NFT has multiple parents, its source rewards are split evenly between all parents
Status: tested and verified ✅

5.2.5 Data NFTs are minted by data users. They pay for the minting and participate by claiming future rewards from the minting pool
Status: tested and verified ✅

5.2.6 The smart contract is coded according to the newest standards and in a secure way
Status: tested and verified ✅

6. Executive Summary

UPDATED Sep. 04.2021

Two (2) independent Chainsulting experts performed an unbiased and isolated audit of the smart contract codebase. The final debriefs took place on the September 04, 2021.

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 critical issues were found after the manual and automated security testing.

7. Deployed Smart Contract

PENDING