SOLIDProof
Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC

MADE IN GERMANY

Audit

Security Assessment
19. January, 2022

For

MASD GAMES

Disclaimer

SolidProof.io reports are not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team. SolidProof.io do not cover testing or auditing the integration with external contract or services (such as Unicrypt, Uniswap, PancakeSwap etc'…)

SolidProof.io Audits do not provide any warranty or guarantee regarding the absolute bug- free nature of the technology analyzed, nor do they provide any indication of the technology proprietors. SolidProof Audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

SolidProof.io Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. Blockchain technology and cryptographic assets present a high level of ongoing risk. SolidProof's position is that each company and individual are responsible for their own due diligence and continuous security. SolidProof in no way claims any guarantee of security or functionality of the technology we agree to analyze.

Version	Date	Description
1.0	19. January 2022	• Layout project • Automated- /Manual-Security Testing • Summary

Network
Binance Smart Chain (BEP20)

Website
https://masd.games/

Telegram
https://t.me/Masd_Games

Twitter
https://twitter.com/Masd_Games

Github
https://github.com/MasdGames

Medium
https://medium.com/@Masd.Games

Discord
https://discord.com/invite/masdgames

Youtube
https://www.youtube.com/watch?v=L_FtIL0Bp9Q

Twitch
https://www.twitch.tv/masd_games

Description

MASD is a multiplayer 3D game with blockchain technology. Be the first to create your base and play to earn a MASD token, buy or rent virtual plots of land, houses, items for survival in the game. Create your own clans and communities around the world in the game to participate in competitions for valuable items and MASD tokens together.
Being inspired by successful games like CS:GO and RUST, MASD-GAMES combines the best of both worlds and also allows players to earn money playing their favorite game and own limited resources purchased in the form of NFT

Project Engagement

During the 14th January of 2022, **MASD Games Team** engaged Solidproof.io to audit smart contracts that they created. The engagement was technical in nature and focused on identifying security flaws in the design and implementation of the contracts. They provided Solidproof.io with access to their code repository and whitepaper.

Logo

Contract Link
v1.0
Github
- https://github.com/MasdGames/masd-games-contracts
- Commit: 7603d8423a5d053db7f451a03295bde78b3ad7ab

Mainnet
- MASD
  1. https://bscscan.com/address/0xfcc92ae68facbdb6372fce8fbcacc08b67f8a744#code
- Vesting
  2. https://bscscan.com/address/0x95a3c00d5d35ac0f125f9782838a086750103c21#code
  3. https://bscscan.com/address/0x517d6b7562ee67eeb55e4561b8865d326695d5db#code
  4. https://bscscan.com/address/0xa1c585cae8c738b9b1ab3ac0bcc8ac25e02bc1e0#code

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 aspossible.
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

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.

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 SolidProof 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 SolidProof 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.

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

Imported packages:

Dependency / Import Path	Count
@openzeppelin/contracts/access/AccessControlEnumerable.sol	1
@openzeppelin/contracts/access/Ownable.sol	1
@openzeppelin/contracts/token/ERC20/IERC20.sol	1
@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol	1
@openzeppelin/contracts/token/ERC20/extensions/ERC20Capped.sol	1
@openzeppelin/contracts/token/ERC20/extensions/draft-ERC20Permit.sol	1
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	1
@openzeppelin/contracts/utils/math/Math.sol	1
@openzeppelin/contracts/utils/math/SafeMath.sol	1

Tested Contract Files

This audit covered the following files listed below with a SHA-1 Hash.

A file with a different Hash has been modified, intentionally or otherwise, after the security review. A different Hash could be (but not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of this review.

v1.0

| File Name	SHA-1 Hash
contracts/libraries/BP.sol	f2b9d1c7959393ffca55f0f17a6d3fd73957a304
contracts/libraries/PercentageVestingLibrary.sol	59e81e6a1a735153375eae65f5d6257ef71e0a6c
contracts/MASD.sol	cadc3214220c383f3bae82d77e463cca275a87d0
contracts/MASDVesting.sol	a8468f5ab1f879640883ad06e2778bca5ac5bb79

Metrics

Source Lines
v1.0
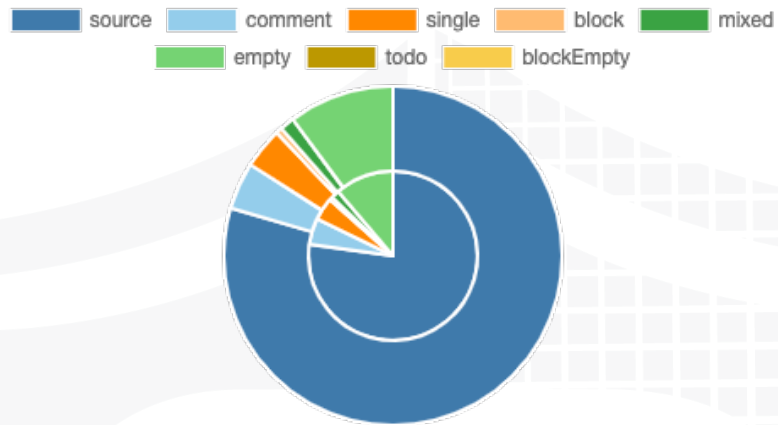
Risk Level
v1.0
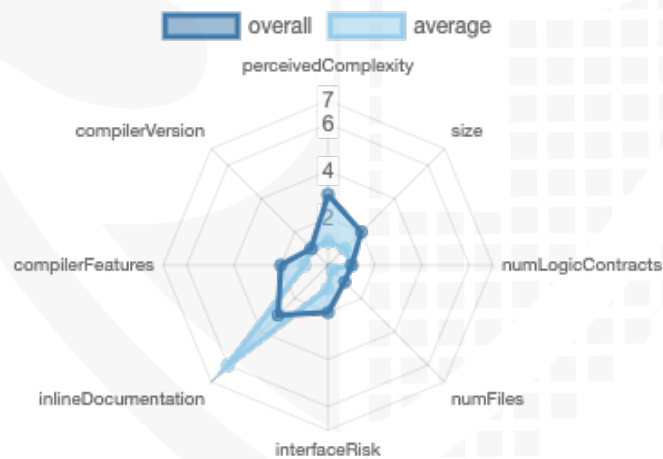
Capabilities

Components

Version	Contracts	Libraries	Interfaces	Abstract
1.0	2	2	0	0

Exposed Functions

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

Version	Public	Payable
1.0	9	0

Version	External	Internal	Private	Pure	View
1.0	7	13	1	0	7

State Variables

Version	Total	Public
1.0	9	9

Capabilities

Version	Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts

1.0	`^0.8.4`
	`^0.8.0`
	`0.8.6`				

Version	Transfers ETH	Low-Level Calls	DelegateCall	Uses Hash Functions	EC Recover	New/ Create/ Create2
1.0				yes		

Inheritance Graph
v1.0

CallGraph
v1.0

Legend

Internal Call	→
External Call	→
Defined Contract	▢
Undefined Contract	▢

MASDVesting

- userVestingsLength
- <Constructor> → IERC20
- getVestingParams
- getWalletInfo → getUserVesting
- createVestingParams
- withdrawAll
- withdraw
- createUserVesting

IERC20

- safeTransfer
- safeTransferFrom

MASD

- <Constructor> → _setupRole
- mint → _mint

ERC20Permit

- _mint

ERC20

- _mint

PercentageVestingLibrary (lib)
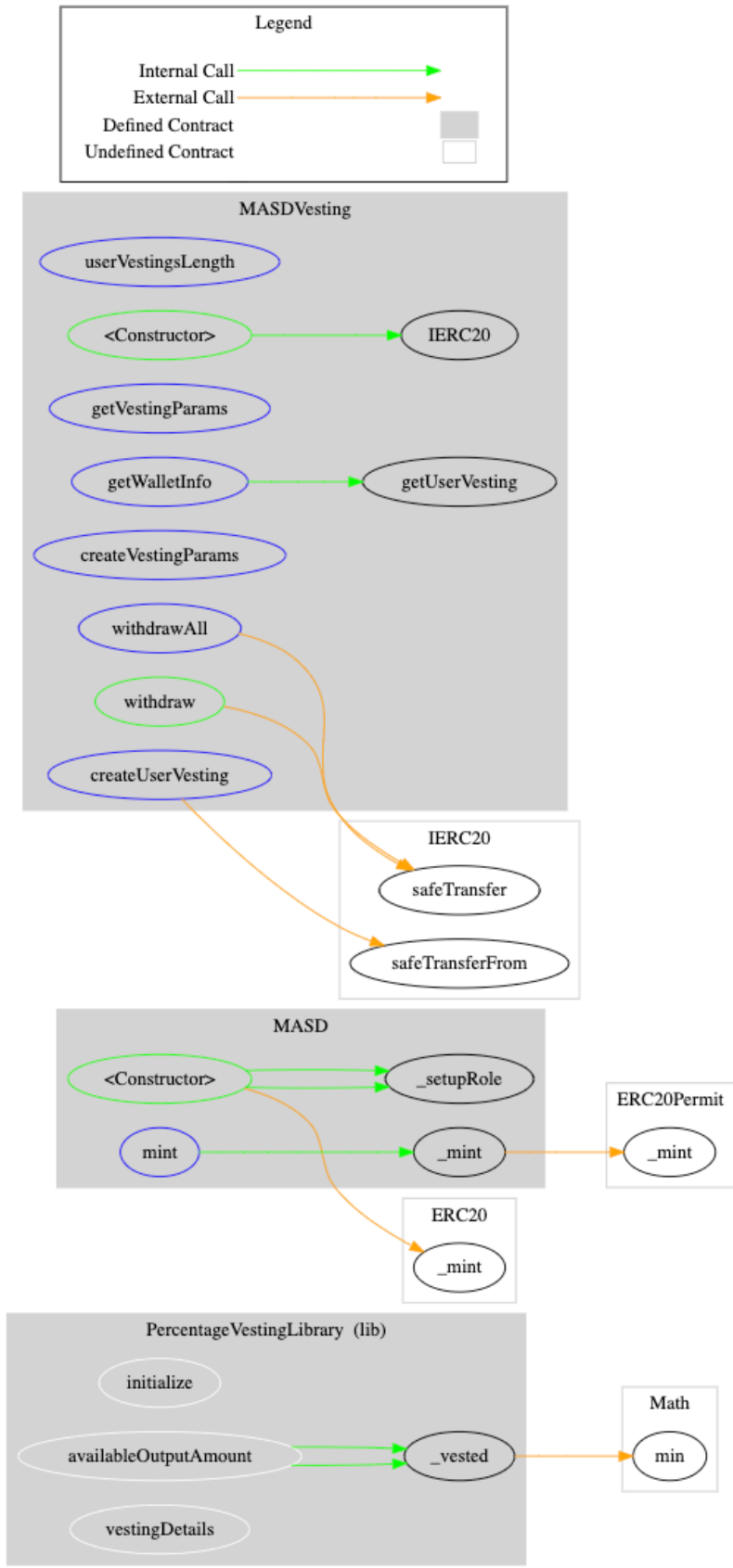
- initialize
- availableOutputAmount → _vested
- vestingDetails

Math

- min

Scope of Work/Verify Claims

The above token Team provided us with the files that needs to be tested (Github, Bscscan, Etherscan, files, etc.). The scope of the audit is the main contract (usual the same name as team appended with .sol).

We will verify the following claims:

1. Correct implementation of Token standard
2. Deployer cannot mint any new tokens
3. Deployer cannot burn or lock user funds
4. Deployer cannot pause the contract
5. External approve function is restricted
6. Overall checkup (Smart Contract Security)

Correct implementation of Token standard

Function	Description	Exist	Tested	Verified
TotalSupply	provides information about the total token supply	✓	✓	✓
BalanceOf	provides account balance of the owner's account	✓	✓	✓
Transfer	executes transfers of a specified number of tokens to a specified address	✓	✓	✓
TransferFrom	executes transfers of a specified number of tokens from a specified address	✓	✓	✓
Approve	allow a spender to withdraw a set number of tokens from a specified account	✓	✓	✓
Allowance	returns a set number of tokens from a spender to the owner	✓	✓	✓

Write functions of contract v1.0

MASD

- approve
- burn
- burnFrom
- decreaseAllowance
- grantRole
- increaseAllowance
- mint
- permit
- renounceRole
- revokeRole
- transfer
- transferFrom

MASDVESTING

- createUserVesting
- createVestingParams
- renounceOwnership
- transferOwnership
- withdraw
- withdrawAll

Deployer cannot mint any new tokens

Name	Exist	Tested	Status
Deployer cannot mint	✓	✓	✓
Max / Total Supply	100.000.000 / can be set while deploying		

Comments:
v1.0
- Only address with MINTER_ROLE can mint new tokens as long as minting amount + total supply is lower equal max_supply.
- **The Minter Role has been revoked**
  - https://bscscan.com/tx/0xb10e0fd59f0211ef6b4dc02db240ff29928936efd843ed8041fa1b7bb95cbf82
  - https://bscscan.com/tx/0x10bc573274353376e1ba73e47ea4f873e9bb0758a9a8fd9e4961e68cfb3f9454
  -

MASD token deployed at 0xfcc92ae68facbDb6372fce8fBCaCC08b67f8A744 has no ability to mint new tokens because ADMIN and MINTER role were revoked.

Deployer cannot burn or lock user funds

Name	Exist	Tested	Status
Deployer cannot lock	✓	✓	✓
Deployer cannot burn	✓	✓	✓

Comments:
v1.0
·   Everyone can burn their own tokens.

Deployer cannot pause the contract

Name	Exist	Tested	Status
Deployer cannot pause	–	–	–

Comments:
v1.0
- No pause function found in the contract.

External approve function is restricted

Name	Exist	Tested	Status
External approve cannot be called without restriction	–	–	–

Comments:

v1.0

- There are no external calls in the contract, so there is no external call without restriction.

Overall checkup (Smart Contract Security)

Tested	Verified
✓	✓

Legend

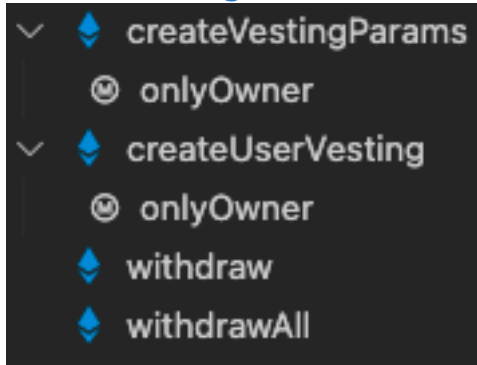| Attribute	Symbol
Verfified / Checked	✓
Partly Verified	🚩
Unverified / Not checked	✘
Not available	–

Modifiers and public functions
## v1.0
MASD
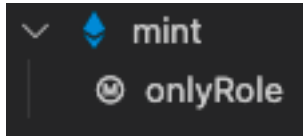
MASDVesting

Please check if an OnlyOwner or similar restrictive modifier has been forgotten.

Source Units in Scope
v1.0

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📚	contracts/libraries/BP.sol	1	———	7	7	4	1	2	———
📚	contracts/libraries/PercentageVestingLibrary.sol	1	———	102	91	74	10	17	———
📝	contracts/MASD.sol	1	———	30	30	23	1	26	🎨
📝	contracts/MASDVesting.sol	1	———	192	166	146	4	67	———
📝📚	**Totals**	**4**	———	**331**	**294**	**247**	**16**	**112**	🎨

Legend

Attribute	Description
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, ...)

Audit Results

Critical issues

No critical issues

High issues

No high issues

Medium issues

No medium issues

Low issues

Issue	File	Type	Line	Description
#1	BP	A floating pragma is set	2	The current pragma Solidity directive is „^0.8.4"".
#2	PercentageVestingLibrary	A floating pragma is set	2	The current pragma Solidity directive is „^0.8.4"".
#3	MASD	A floating pragma is set	2	The current pragma Solidity directive is „^0.8.0"".

Informational issues

Issue	File	Type	Line	Description
#1	Main	NatSpec documentation missing	-	If you start to comment your code, also comment all other functions, variables etc.

24

#2	Main	SafeMath library is not required	-	SafeMath is not required above pragma version 0.8.x. If you remove SafeMath, make sure to remove all SafeMath
#3	Main	Unused variables	96, 99	Remove unused variables
#4	PercentageVestingLibrary	Modifier ordering	47, 83,	Following visibility modifier should come before other modifiers - Internal - Private
#5	MASDVesting	Misspelling	72, 81, 100, 104	Change - avaliable to available Make sure to change it everywhere else also

Audit Comments

We recommend you to use the special form of comments (NatSpec Format, Follow link for more information https://docs.soliditylang.org/en/v0.5.10/natspec-format.html) for your contracts to provide rich documentation for functions, return variables and more. This helps investors to make clear what that variables, functions etc. do.

19. January 2022:

- Read whole report for more information
- MASD token deployed at 0xfcc92ae68facbDb6372fce8fBCaCC08b67f8A744 was compiled using exact 0.8.6 solc version.

SWC Attacks

ID	Title	Relationships	Status
SWC-136	Unencrypted Private Data On-Chain	CWE-767: Access to Critical Private Variable via Public Method	PASSED
SWC-135	Code With No Effects	CWE-1164: Irrelevant Code	PASSED
SWC-134	Message call with hardcoded gas amount	CWE-655: Improper Initialization	PASSED
SWC-133	Hash Collisions With Multiple Variable Length Arguments	CWE-294: Authentication Bypass by Capture-replay	PASSED
SWC-132	Unexpected Ether balance	CWE-667: Improper Locking	PASSED
SWC-131	Presence of unused variables	CWE-1164: Irrelevant Code	PASSED
SWC-130	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	PASSED
SWC-129	Typographical Error	CWE-480: Use of Incorrect Operator	PASSED
SWC-128	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	PASSED

[SWC-127](#)	Arbitrary Jump with Function Type Variable	[CWE-695: Use of Low-Level Functionality](#)	**PASSED**
[SWC-125](#)	Incorrect Inheritance Order	[CWE-696: Incorrect Behavior Order](#)	**PASSED**
[SWC-124](#)	Write to Arbitrary Storage Location	[CWE-123: Write-what-where Condition](#)	**PASSED**
[SWC-123](#)	Requirement Violation	[CWE-573: Improper Following of Specification by Caller](#)	**PASSED**
[SWC-122](#)	Lack of Proper Signature Verification	[CWE-345: Insufficient Verification of Data Authenticity](#)	**PASSED**
[SWC-121](#)	Missing Protection against Signature Replay Attacks	[CWE-347: Improper Verification of Cryptographic Signature](#)	**PASSED**
[SWC-120](#)	Weak Sources of Randomness from Chain Attributes	[CWE-330: Use of Insufficiently Random Values](#)	**PASSED**
[SWC-119](#)	Shadowing State Variables	[CWE-710: Improper Adherence to Coding Standards](#)	**PASSED**
[SWC-118](#)	Incorrect Constructor Name	[CWE-665: Improper Initialization](#)	**PASSED**
[SWC-117](#)	Signature Malleability	[CWE-347: Improper Verification of Cryptographic Signature](#)	**PASSED**

SWC-116	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	**PASSED**
SWC-115	Authorization through tx.origin	CWE-477: Use of Obsolete Function	**PASSED**
SWC-114	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	**PASSED**
SWC-113	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	**PASSED**
SWC-112	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	**PASSED**
SWC-111	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	**PASSED**
SWC-110	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	**PASSED**
SWC-109	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	**PASSED**
SWC-108	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	**PASSED**
SWC-107	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	**PASSED**
SWC-106	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	**PASSED**

SWC-105	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	**PASSED**
SWC-104	Unchecked Call Return Value	CWE-252: Unchecked Return Value	**PASSED**
SWC-103	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	**NOT PASSED**
SWC-102	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	**PASSED**
SWC-101	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	**PASSED**
SWC-100	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	**PASSED**

Solid Proofed

Blockchain Security | Smart Contract Audits | KYC

MADE IN GERMANY