SOLIDProof
Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC

MADE IN GERMANY

MetaverseDAO

Audit

Security Assessment

30. March, 2022

For

Metaverse DAO

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	30. March 2022	• Layout project • Automated- /Manual-Security Testing • Summary

Network
Binance Smart Chain (BEP20)

Website
https://www.metaverse-dao.io/

Telegram
https://t.me/metaverse_dao_community

Twitter
https://twitter.com/METADAO_Offical

Facebook
https://www.facebook.com/Metaverse-DAO-107675525207534

Reddit
https://www.reddit.com/r/Metaverse_DAO

Medium
https://medium.com/@Metaverse-DAO

Youtube
https://www.youtube.com/watch?v=SEHWStUUD9g

Description

Metaverse-DAO (METADAO) is a decentralized autonomous organization (DAO) for the development, management of decentralized network based games, NFTs, finance (Defi) and other projects.

Our mission is to build strong virtual worlds (part of the metaverse), create really fun games and Dapps, and manage all the projects and assets with our community members, allowing token holders to share the profits.

Project Engagement

During the 27th of March 2022, **MetaverseDAO 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

- https://bscscan.com/address/0xE7697b4965820E4fFdDfE218A2b558166Fba351B#writeContract

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:

```
Address.sol
Context.sol
DividendPayingToken.sol
DividendTracker.sol
ERC20.sol
IDividendPayingToken.sol
IDividendPayingTokenOptional.sol
IERC20.sol
IterableMapping.sol
IUniswapV2Factory.sol
IUniswapV2Pair.sol
IUniswapV2Router01.sol
IUniswapV2Router02.sol
Math.sol
metaversedao.sol
Ownable.sol
SafeMath.sol
SafeMathInt.sol
SafeMathUint.sol
```

Tested Contract Files

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

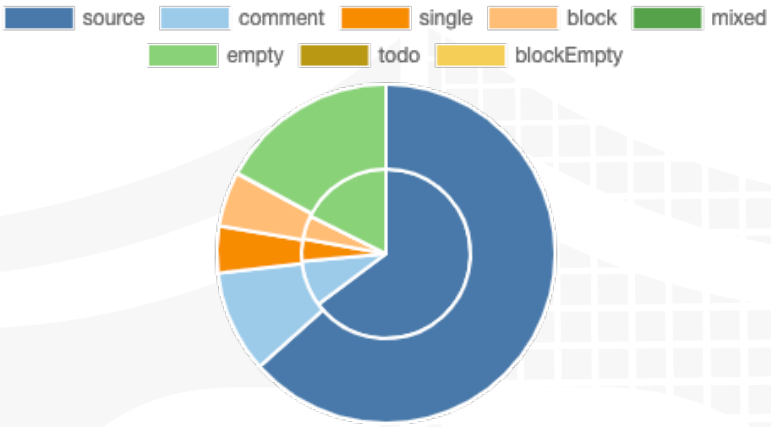
A file with a different Hash has been modified, intentionally or otherwise, after the security review. A different Hash could be (but not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of this review.

v1.0

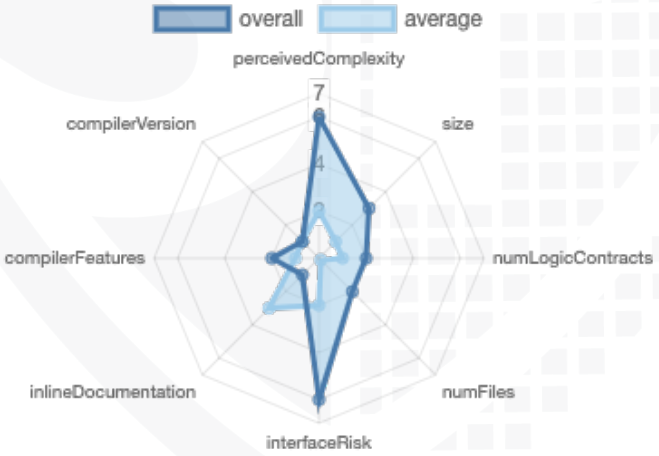
File Name	SHA-1 Hash
contracts/DividendPayingToken.sol	9fa1f87a7a39c81e7c75f1dfea6975f2f1f8848f
contracts/IUniswapV2Pair.sol	93e24ad0ad08c5480ffa45dba9cb55b7203688f4
contracts/SafeMathUint.sol	519fd1937a3d76270282d3ccb9ec7163992708aa
contracts/Math.sol	f6455bdcf310b73ce5fb9d6a6035ba0512e9cf6c
contracts/Context.sol	15561b12740d13715a13ca534aa6036c1227d8c0
contracts/IUniswapV2Factory.sol	a5a1c7cf581ea1fcdda5fb226eaa494c69e1df11
contracts/SafeMathInt.sol	300538bf12abba4c0bea23ead3079bbb43e27e49
contracts/IDividendPayingTokenOptional.sol	3031c68dfb51a079268098ce88e70b728bb55889
contracts/Address.sol	cd7032c7c90284aa2c03fd6fe1d958297c2a928f
contracts/SafeMath.sol	f04c94c0600e0edb6f63fbd3861c74722094401e
contracts/IDividendPayingToken.sol	521bae9924be430526a46ea2f7abda0ac2410a2a
contracts/Ownable.sol	b2d7d4c0ec2a1b3901a1d9e75d34b85c8b1ef67f
contracts/IterableMapping.sol	a6754b03427b59bbe7dccb7f6fc52ff22bbdee72
contracts/IUniswapV2Router02.sol	ceeb197905988a225c220f9cb959571be4a2140b
contracts/IUniswapV2Router01.sol	f5eb22b303c24a061253d191fe59e32486c1fe66
contracts/ERC20.sol	906459a5d75b875289f06bc26fbd9a1376ddf391
contracts/DividendTracker.sol	787d7f648818c47d5fe54cc3b45b263065edd289
contracts/metaversedao.sol	9efaf281603b4db22ba5cc211cbd8923c3637bef
contracts/IERC20.sol	124d442a8a0abcba5afbc2ed8b5a10ec2292c5e3

Metrics

Source Lines
v1.0



Risk Level
v1.0



Capabilities

Components

Version	Contracts	Libraries	Interfaces	Abstract
1.0	5	6	7	1

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	165	8

Version	External	Internal	Private	Pure	View
1.0	79	151	15	27	78

State Variables

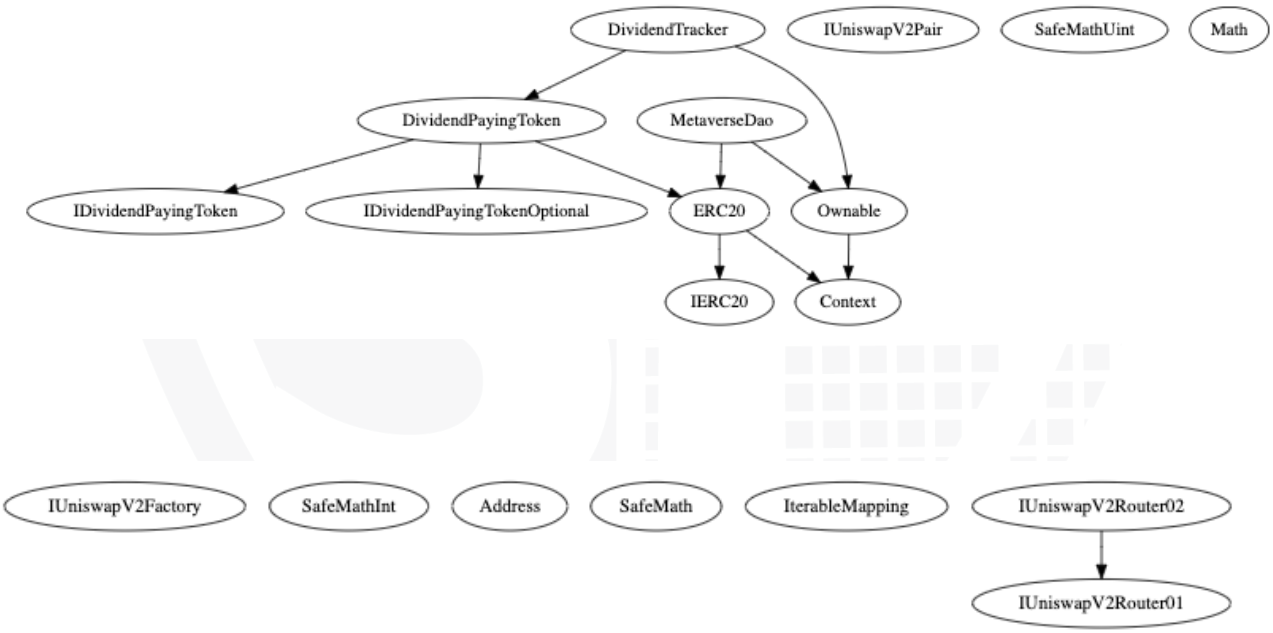
Version	Total	Public
1.0	62	39

Capabilities

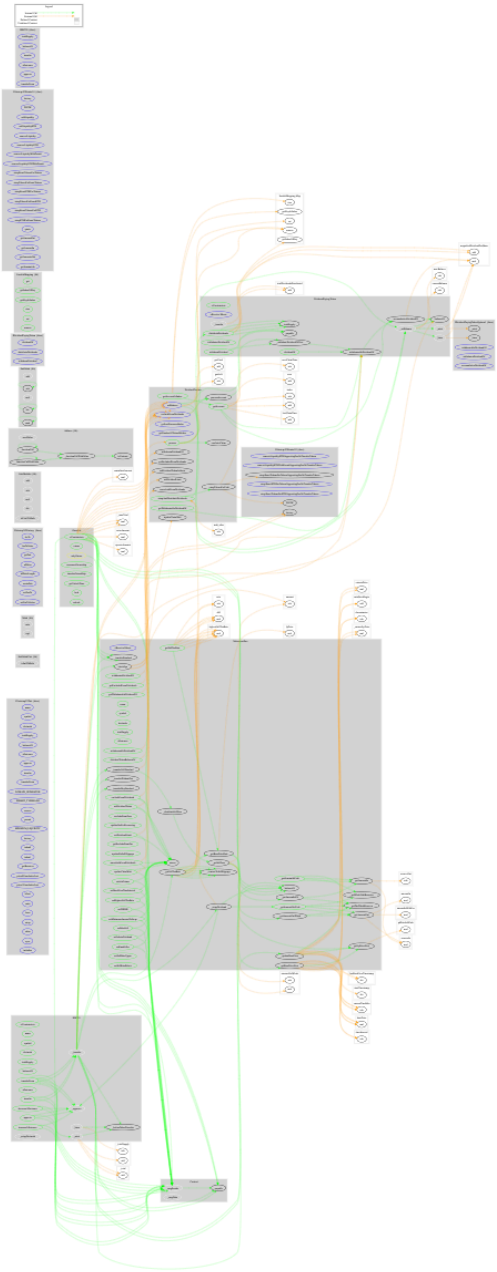
Version	Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
1.0	`^0.6.12`	`ABIEncoderV2`	`yes`	`yes` (2 asm blocks)	

Version	Transfers ETH	Low-Level Calls	DelegateCall	Uses Hash Functions	ECRecover	New/ Create/ Create2
1.0	`yes`					`yes → NewContract:DividendTracker`

Inheritance Graph
v1.0



CallGraph
v1.0



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. Overall checkup (Smart Contract Security)

Correct implementation of Token standard

ERC20				
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

1. approve
2. decreaseAllowance
3. excludeFromDividends
4. excludeFromFees
5. increaseAllowance
6. lock
7. renounceOwnership
8. setBasePriceTimeInterval
9. setDividendLimit
10. setDividendToken
11. setEnableFee
12. setHighestSellTaxRate
13. setIsAutoDividend
14. setMaxSell
15. setMinimumAmountToSwap
16. setSellRateBelow
17. setSellRateUpper
18. setWBNB
19. switchOwner
20. transfer
21. transferFrom
22. transferOwnerShip
23. unexcludeFromDividends
24. unlock
25. updateClaimWait
26. updateFixSellSlippage
27. updateGasForProcessing

Deployer cannot mint any new tokens

Name	Exist	Tested	Status
Deployer cannot mint	✓	✓	✓
Max / Total Supply			350.000.000

Deployer cannot burn or lock user funds

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

Comments:

v1.0

- _setBalance in DividendPayingToken will mint/burn tokens but deployer cannot burn tokens for specific addresses
- Deployer can lock user funds by
  - Setting max sell variable to 0

Deployer cannot pause the contract

Name	Exist	Tested	Status
Deployer cannot pause	–	–	–

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

- transfer
- transferFrom
- approve
- switchOwner
- setBasePriceTimeInterval
- setHighestSellTaxRate
- setWBNB
- setMinimumAmountToSwap
- setMaxSell
- setIsAutoDividend
- excludeFromFees
  - onlyOwner
- updateGasForProcessing
  - onlyOwner
- setEnableFee
- updateFixSellSlippage
  - onlyOwner
- setDividendLimit
- setDividendToken
- updateClaimWait
- excludeFromDividends
- unexcludeFromDividends
- setSellRateUpper
  - onlyOwner
- setSellRateBelow
  - onlyOwner

- renounceOwnership
  - onlyOwner
- transferOwnerShip
  - onlyOwner
- lock
  - onlyOwner
- unlock

Comments

- Deployer can set following state variables without any limitations
  - basePriceTimeInterval
  - highestSellTaxRate
  - minimumAmountToSwap
  - _maxSell
  - fixSellSlippage
  - _dividendLimitUsd
  - sellRateUpper
  - sellRateBelow

- Deployer can enable/disable following state variables
  - isAutoDividend
  - _isExcludedFromFee
  - enableFee
  - excludedFromDividends

- Deployer can set following addresses
  - _owner
  - WBNB
  - dividendToken

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/DividendPayingToken.sol	1	————	199	199	101	53	94	🖊...💰.💠
🔍	contracts/IUniswapV2Pair.sol	————	1	74	9	6	1	55	🖊..
📚	contracts/SafeMathUint.sol	1	————	11	11	9	1	3	🖊..
📚	contracts/Math.sol	1	————	25	25	19	3	5	🖊..
🎨	contracts/Context.sol	1	————	13	13	10	2	2	————
🔍	contracts/IUniswapV2Factory.sol	————	1	19	8	5	1	17	🖊..
📚	contracts/SafeMathInt.sol	1	————	33	33	27	1	8	🖊..
🔍	contracts/IDividendPayingTokenOptional.sol	————	1	27	15	4	16	7	🖊..
📚	contracts/Address.sol	1	————	76	64	38	18	37	💻🖊..
📚	contracts/SafeMath.sol	1	————	65	65	40	14	10	🖊..
🔍	contracts/IDividendPayingToken.sol	————	1	43	15	4	22	10	🖊...💰
📝	contracts/Ownable.sol	1	————	70	70	45	12	40	🖊..
📚	contracts/IterableMapping.sol	1	————	64	64	50	2	19	🖊..
🔍	contracts/IUniswapV2Router02.sol	————	1	50	9	5	1	16	🖊...💰
🔍	contracts/IUniswapV2Router01.sol	————	1	97	6	4	1	48	🖊...💰
📝	contracts/ERC20.sol	1	————	138	138	89	25	79	🖊..
📝	contracts/DividendTracker.sol	1	————	274	256	184	1	141	🖊..
📝	contracts/metaversedao.sol	1	————	658	658	540	3	544	🖊...💰.🌀.♻️
🔍	contracts/IERC20.sol	————	1	60	18	12	38	13	🖊...🔆
📝📚🔍🎨	**Totals**	**12**	**7**	**1996**	**1676**	**1192**	**215**	**1148**	💻🖊...💰.💠🌀.🔆.♻️

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

AUDIT PASSED

Critical issues

No critical issues

High issues

No high issues

Medium issues

No medium issues

Low issues

Issue	File	Type	Line	Description
#1	Main	Contract doesn't import npm packages from source (like OpenZeppelin etc.)	-	We recommend to import all packages from npm directly without flatten the contract. Functions could be modified or can be susceptible to vulnerabilities
#2	All	A floating pragma is set	At the top of source files	The current pragma Solidity directive is „"^0.6.12"".
#3	DividendTracker	Missing Zero Address Validation (missing-zero-check)	240	Check that the address is not zero
#4	MetaverseDao	State variables shadowing	20, 22, 37, 27, 23, 28	Rename the state variables that shadow another component
#5	MetaverseDao	Local variables shadowing	176, 520	Rename the local variables that shadow another component

#6	DividendPayingToken	Local variables shadowing	57	Rename the local variables that shadow another component
#7	MetaverseDao	Missing Events Arithmetic	640, 636, 587	Emit an event for critical parameter changes

Informational issues

Issue	File	Type	Line	Description
#1	DividendPayingToken	State variables that could be declared constant (constable-states)	33	Add the `constant` attributes to state variables that never change
#2	MetaverseDao	State variables that could be declared constant (constable-states)	55, 37, 54, 33, 27, 56, 57, 28, 45, 29, 30	Add the `constant` attributes to state variables that never change
#3	DividendPayingToken	Unused state variables	33	Remove unused state variables
#4	MetaverseDao	Misspelling	See description	Change following words: - reciever to receiver L128 Make sure to change it everywhere else as well.

#5	Dividen dPaying Token	Unreachable code	156-162	Remove the following coloured part from function because it will never executed:
				
				function _transfer(address from, address to, uint256 value) internal virtual override {
				require(false);
				
				int256 _magCorrection = magnifiedDividendPerShare.mul(value).toInt256Safe();
				
				magnifiedDividendCorrections[from] = magnifiedDividendCorrections[from].add(_magCorrection);
				
				magnifiedDividendCorrections[to] = magnifiedDividendCorrections[to].sub(_magCorrection);
				}

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.

30. March 2022:

- Read whole report for more information

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	NOT 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	NOT 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	NOT 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