SOLIDProof
Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC

MADE IN GERMANY

Consensus

Audit

Security Assessment

17. March, 2022

For

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

Network

Polygon

Website

https://www.synassets.finance/#/

Twitter

https://twitter.com/SynAssets

Github

https://github.com/synassets

Medium

https://medium.com/@SynAssets

Discord

https://discord.com/invite/BzVUJJszaU

Description

TBA

Project Engagement

During the 14th of March 2022, **Consensus 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/synassets/contracts/blob/main/contracts/TokenSale.sol
  - Commit: 8d1dee52e74a0e487c657918913284055bda4b8e

  - https://github.com/synassets/contracts/blob/main/contracts/ConsensusPool.sol
  - Commit: e8b24a2693e73b41cb7b5130aecf21ec4dae2272

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.

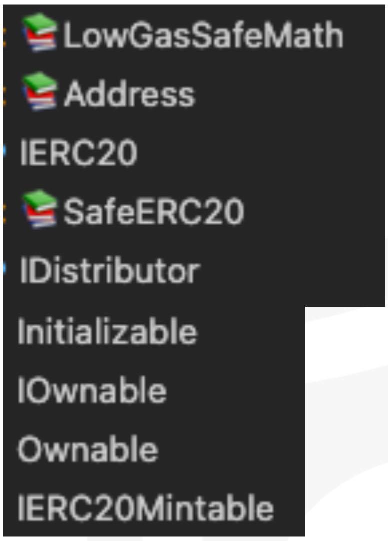
2. Testing and automated analysis that includes the following:
    i) Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
    ii) Symbolic execution, which is analysing a program 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:



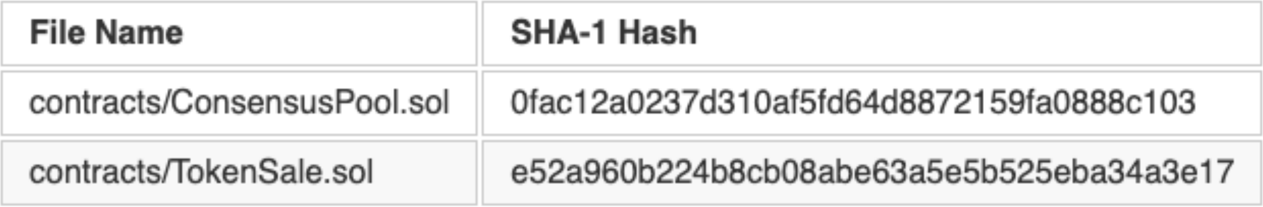
LowGasSafeMath
Address
IERC20
SafeERC20
IDistributor
Initializable
IOwnable
Ownable
IERC20Mintable

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

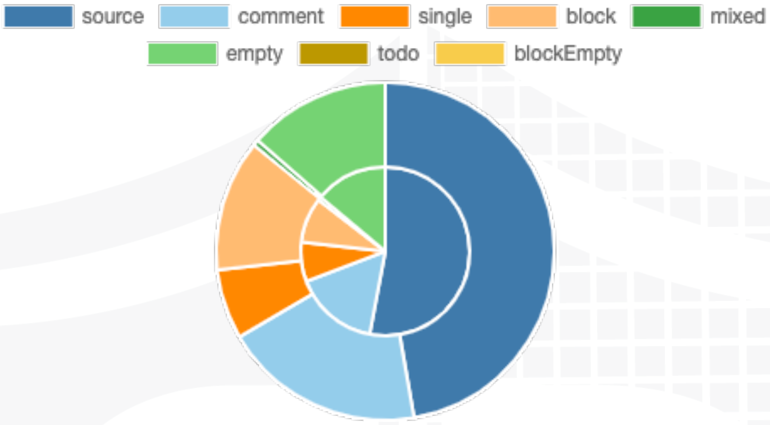


File Name	SHA-1 Hash
contracts/ConsensusPool.sol	0fac12a0237d310af5fd64d8872159fa0888c103
contracts/TokenSale.sol	e52a960b224b8cb08abe63a5e5b525eba34a3e17

Metrics

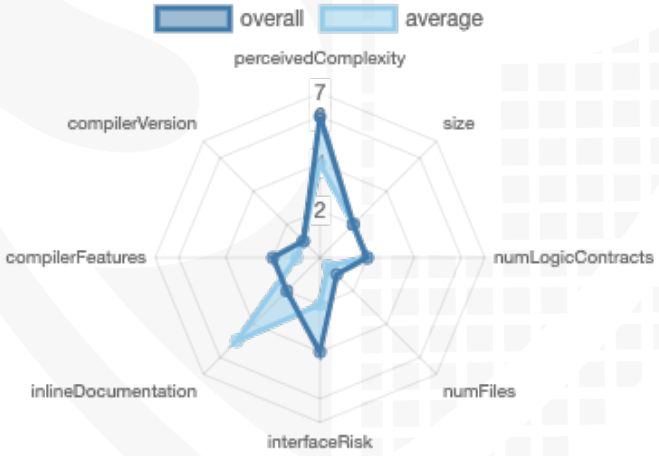
Source Lines

v1.0



Risk Level

v1.0



Capabilities

Components

Version	Contracts	Libraries	Interfaces	Abstract
1.0	4	6	5	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	37	1

Version	External	Internal	Private	Pure	View
1.0	32	92	7	20	17

State Variables

Version	Total	Public
1.0	52	47

Capabilities

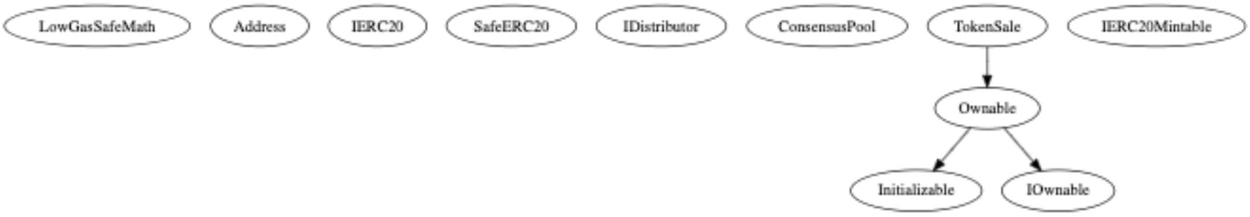
Version	Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
1.0	`0.7.5`		`yes`	`yes` (7 asm blocks)	

Version	Transfers ETH	Low-Level Calls	DelegateCall	Uses Hash Functions	ECRecover	New/Create/Create2

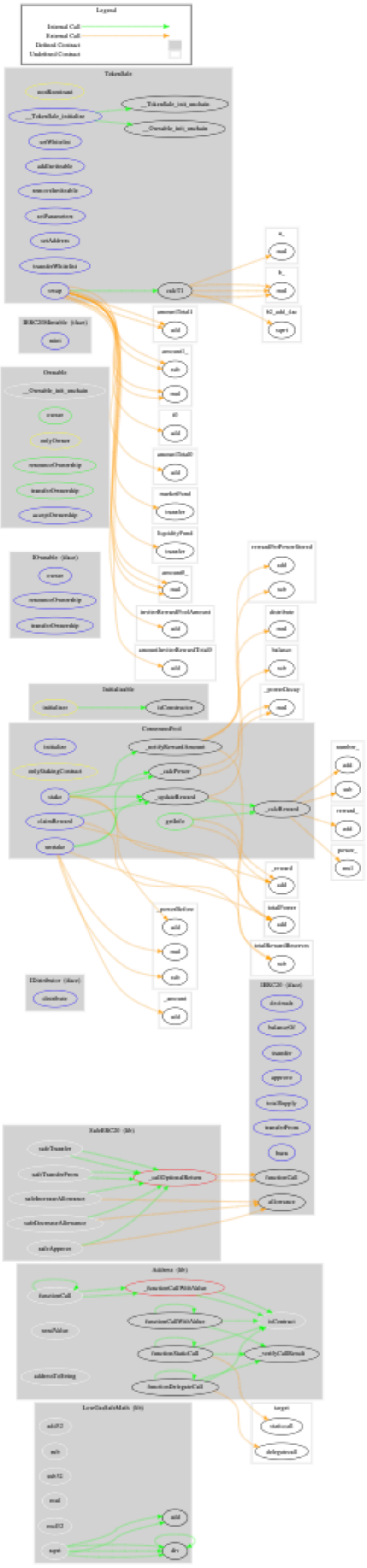
1.0	yes		yes			

Inheritance Graph

v1.0



CallGraph
v1.0



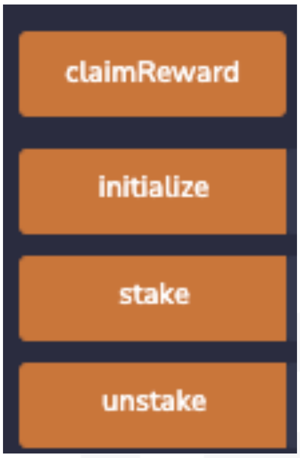
Scope of Work/Verify Claims

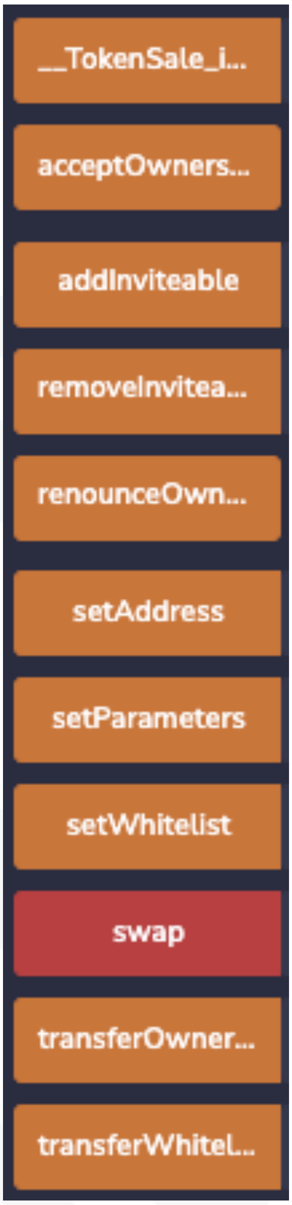
The above token Team provided us with the files that needs to be tested (Github, Bscscan, Etherscan, files, etc.). The scope of the audit is the main contract (usual the same name as team appended with .sol).

We will verify the following claims:

1. Overall checkup (Smart Contract Security)

Write functions of contract v1.0





Overall checkup (Smart Contract Security)

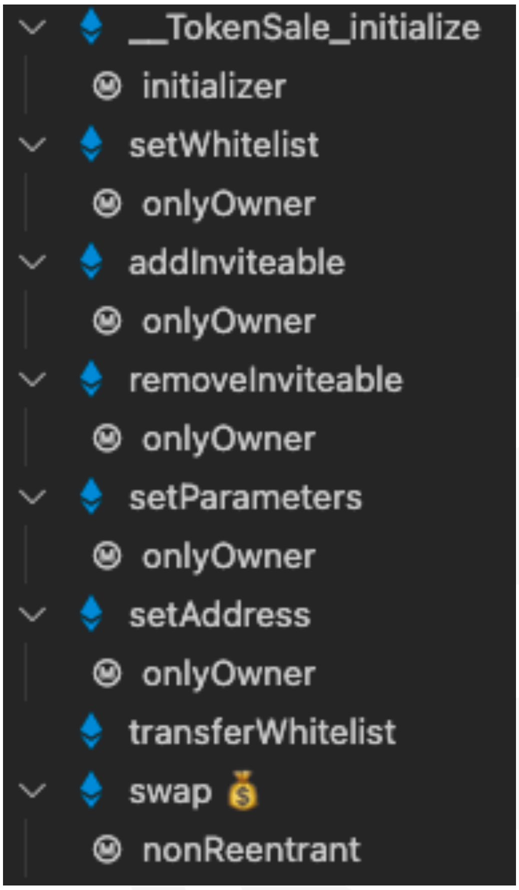
Tested	Verified
✓	✓

Legend

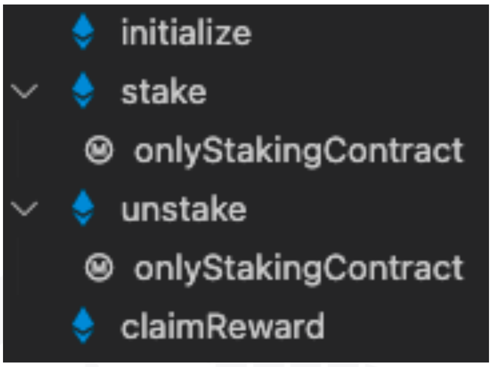
Attribute	Symbol
Verfified / Checked	✓
Partly Verified	🚩
Unverified / Not checked	✗
Not available	–

Modifiers and public functions

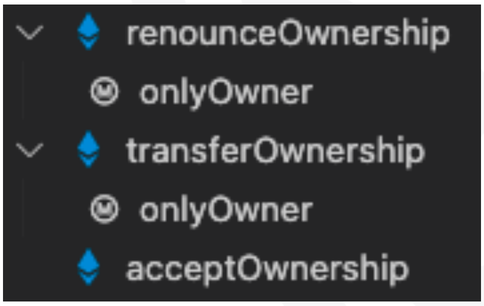
v1.0



```
__TokenSale_initialize
    initializer
setWhitelist
    onlyOwner
addInviteable
    onlyOwner
removeInviteable
    onlyOwner
setParameters
    onlyOwner
setAddress
    onlyOwner
transferWhitelist
swap 💰
    nonReentrant
```



```
initialize
stake
    onlyStakingContract
unstake
    onlyStakingContract
claimReward
```



```
renounceOwnership
    onlyOwner
transferOwnership
    onlyOwner
acceptOwnership
```

Comments

- Deployer can set following state variables without any limitations
    - k
    - kDenominator
    - b
    - bDenominator
    - openAt
    - closeAt
    - maxAmount1
    - maxAmount1PerWallet
    - minAmount1PerWallet
    - ratioInviterReward
    - ratioInviteeReward
    - ratioInviterRewardPool

- Deployer can enable/disable following state variables
  - whitelist
  - inviteable
  - enableWhiteList

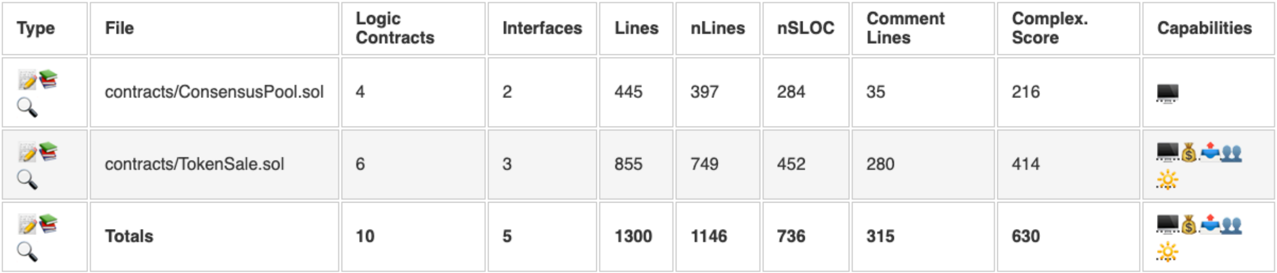
- Deployer can set following addresses
  - marketFund
  - liquidityFund
  - inviterRewardPoolAddress

- Deployer can lock following functions
  - swap
    - By enabling whiteList
    - By setting openAt/closeAt to a high/low value because there is no limitation between openAt <= block.timestamp <= closeAt while setting
    - By setting maxAmount1 to 0 while initializing. This can be updated by the owner. Same for maxAmount1PerWallet

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/ConsensusPool.sol	4	2	445	397	284	35	216	🖥️
📝📚 🔍	contracts/TokenSale.sol	6	3	855	749	452	280	414	🖥️💰📥👥 ☀️
📝📚 🔍	**Totals**	**10**	**5**	**1300**	**1146**	**736**	**315**	**630**	🖥️💰📥👥 ☀️

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	ConsensusPool	Missing Zero Address Validation (missing-zero-check)	244, 243, 272, 294, 337,	Check that the address is not zero
#3	TokenSale	Missing Zero Address Validation (missing-zero-check)	642, 643, 669, 670, 644, 671, 680, 694, 704, 768	Check that the address is not zero
#4	TokenSale	Missing Events Arithmetic	716-727	Emit an event for critical parameter changes
#5	Main	Address cannot be updated	662	Token1 cannot be updated after initializing

#6	TokenSale	Check for zero address	740	It is possible to set following state variables accidentally as zero address: - marketFund - liquidityFund - inviterRewardPoolAddress This can affect to the contract flow and can throw an exception in swap function in L816-L822, L835 We recommend to split the setAddress function into 2 separate functions, one for setAddress and the other for setEnableWhiteList because the setAddress function is only for setting new address, and not bool state variables. Then you can check for zero address in the setAddress function to solve the problem above.

Informational issues

Issue	File	Type	Line	Description
#1	TokenSale	Unused state variables	593	Remove unused state variables
#2	TokenSale	Unused return values	389	Ensure that all the return values of the function calls are used and handle both success and failure cases if needed by the business logic
#3	Main	NatSpec documentation missing	-	If you started to comment your code, also comment all other functions, variables etc.
#4	TokenSale	Rename function variables	843-853	We recommend that you use more descriptive words as variable names.
#5	ConsensusPool	Error message is missing	251, 248, SafeMath require statements	Provide an error message for require statement to inform user which error was thrown

#6	TokenSale	Error message is missing	655, 651, 614, 510, 401, LowGasSafeMath require statements	Provide an error message for require statement to inform user which error was thrown
#7	TokenSale	Misspelling	See description	Change following words and make sure to change it everywhere else as well: - ammount_ to amount_ L545 - dont to don't L760

Commented Code exist

There are some instances of code being commented out in the following files that should be removed:

File	Line	Comment
TokenSale	661	// require(token1_ != address(0), 'IA');
	787	// require(minAmount1PerWallet <= amount1_, 'too few'); fix the second time to take
	777	/* && sender != inviter_*/

Recommendation

Remove the commented code, or address them properly.

Audit Comments

17. March 2022:

- We recommend to use more meaningful words variables for better understandable
- 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	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	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	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