



SOLIDProof
Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC

MADE IN GERMANY

v1.0: 13. January, 2022

Audit

Security Assessment
15. January, 2022

For



VERSAL NFT
stay in eternity

Disclaimer	3
Description	5
Project Engagement	5
Logo	5
Contract Link	5
Methodology	8
Used Code from other Frameworks/Smart Contracts (direct imports)	9
Tested Contract Files	10
Source Lines	11
Risk Level	11
Capabilities	12
Scope of Work	14
Inheritance Graph	14
Verify Claims	15
Modifiers	21
CallGraph	23
Source Units in Scope	24
Critical issues	25
High issues	25
Medium issues	25
Low issues	25
Informational issues	25
Commented Code exist	26
Audit Comments	26
SWC Attacks	27

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	15. January 2022	<ul style="list-style-type: none">• Layout project• Automated- /Manual-Security Testing• Summary
1.1	16. January 2022	<ul style="list-style-type: none">• Reaudit

Network

Binance Smart Chain (BEP20)

Website

<https://versalnft.com/>

Telegram

<https://t.me/versalnft>

https://t.me/versalnft_chat

Twitter

<https://twitter.com/VersalNFT>

Github

<https://github.com/versalnft/smart-contracts>

Reddit

<https://www.reddit.com/user/VersalNFT>

Medium

<https://medium.com/@versalnft>

Description

VersalNFT is a blockchain-based virtual legal space that contains a multi-user interface for creating, storing, and managing data. The basic function of the project is the ability to create a personal digital signature in NFT, containing information about the owner.

Versals (signature creators) will be able to sign documents for business or personal with its help. These documents, in turn, are minted into tokens and immortalized in the blockchain, and stored in crypto wallets. Information about the creator, signers, time, content is recorded in the token and protected from various kinds of manipulation. Using the Unlock protocol, access to content is provided only to signers or a limited number of persons.

The project has a set of rules that are consistent with English legal law. Thus, VersalNFT, using blockchain technology, provides the community with a connection between the crypto space and legal standards in the real world.

Project Engagement

During the 13th of January 2022, **VersalNFT 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

- Testnet
 - VersalToken
 - <https://testnet.bscscan.com/address/0x0cbF2c0554fcBB527c27B19353f49A562dCAbcbE#code>
 - Vest
 - <https://testnet.bscscan.com/address/0x89f4d53f0486401bc8c97EE9F9aaFdb5F16bf6B9#code>

v1.1

- Testnet
 - VersalToken
 - <https://testnet.bscscan.com/address/0x726A0e3293871a6571A30311B06edB0103c1A4A2#code>
 - Vest
 - <https://testnet.bscscan.com/address/0x20818728cA827C7d910b2a270c58F2C913235798#code>
- Mainnet
 - VersalToken
 - <https://bscscan.com/address/0xEbD4F823B4B22c631b1Eb894f46e772B0385c948#code>
 - Vest
 - <https://bscscan.com/address/0x96F7Bc6f91D5229575E1286c1C589B56Cfe86de8#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 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

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:

v1.0

VersalToken



- Context
- Ownable
- IBEP20
-  SafeMath
-  Address
- BEP20
- Initializable

Vest

- Context
- Ownable
- IBEP20

v1.1

Vest

- Context
- Ownable
- IBEP20
-  Address
-  SafeBEP20

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

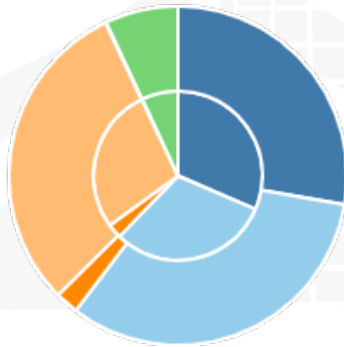
File Name	SHA-1 Hash
contracts/Vest.sol	c72ccc95cdb336ef77c4f9cdf5dd117ef4b566d8
contracts/VersalToken.sol	69ace2432933880913a1652cc35f01affb2f88db

v1.1

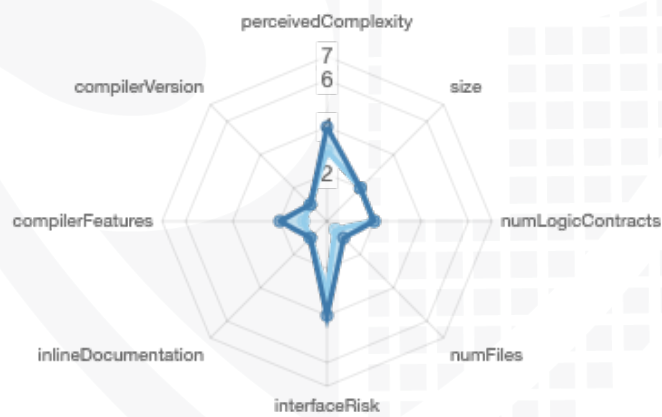
File Name	SHA-1 Hash
contracts/Vest.sol	3090b32bcc7fee60e7ebb22811ca4991b18db0d7
contracts/VersalToken.sol	b7392404e5708427db4a13479c2ad746eb0b9eae

Metrics

Source Lines v1.0



Risk Level v1.0



Capabilities

Components

Version	Contracts	Libraries	Interfaces	Abstract
1.0	7	2	2	1
1.1	7	4	2	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	48	0

Version	External	Internal	Private	Pure	View
1.0	21	70	2	10	29
1.1	21	97	3	11	32

State Variables

Version	Total	Public
1.0	38	28

Capabilities

Version	Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
1.0	0.8.4 0.6.12			yes (3 asm blocks)	

Transfers ETH	Low-Level Calls	DelegateCall	Uses Hash Functions	ERecover	New/Create/Create2
yes		yes			



Scope of Work

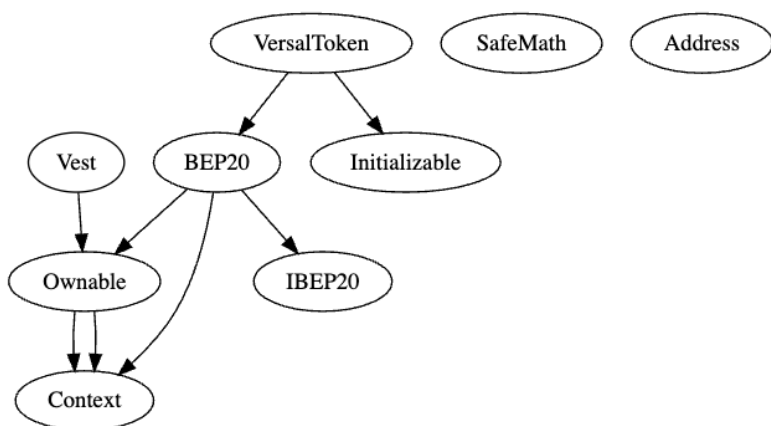
The above token Team provided us with the files that needs to be tested (Github, Bscscan, Etherscan, files, etc.). The scope of the audit is the main contract (usual the same name as team appended with .sol).

We will verify the following claims:

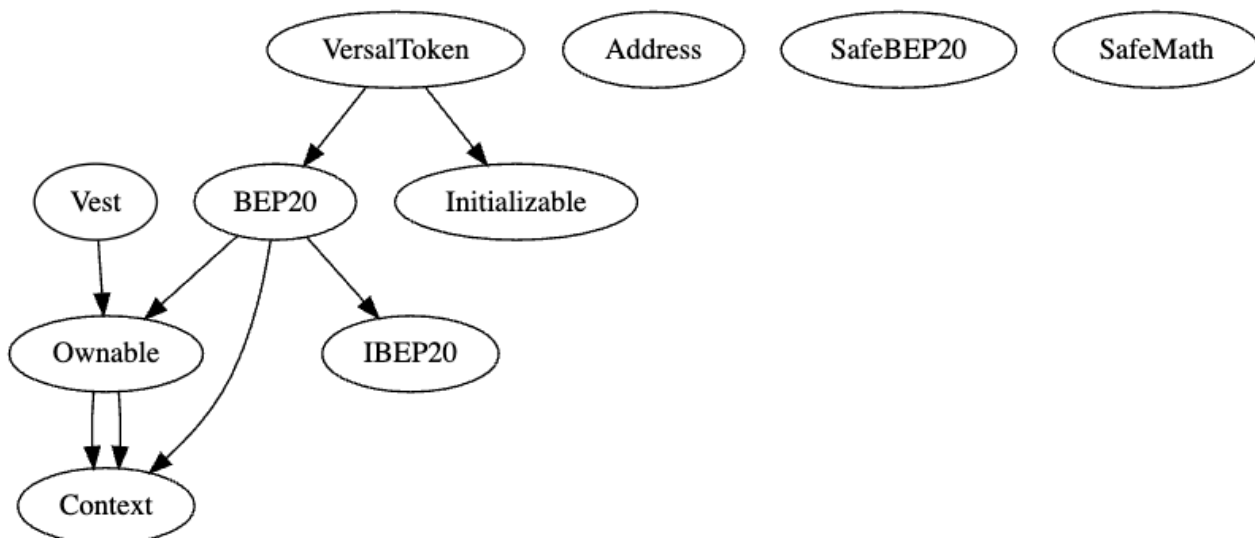
1. Correct implementation of Token standard
2. Deployer cannot mint any new tokens
3. Deployer cannot burn or lock user funds
4. Deployer cannot pause the contract
5. Overall checkup (Smart Contract Security)

Inheritance Graph

v1.0



v1.1



Verify Claims

Correct implementation of Token standard

Tested	Verified
✓	✓

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

VersalToken

Vest

1. approve

2. decreaseAllowance

3. increaseAllowance

4. renounceOwnership

5. setVest

6. transact

7. transfer

8. transferFrom

9. transferOwnership

10. updateWallet

1. addPrivateWallet

2. claimPrivate

3. claimSeed

4. claimTeam

5. contractLock

6. initialize

7. renounceOwnership

8. transferOwnership

Deployer cannot mint any new tokens

Name	Exist	Tested	Verified
Deployer cannot mint	✓	✓	✗

Max / Total Supply: -

Comments:

v1.0

- Deployer can mint with transact function
 - If function called a percentage of the amount is sent out to addresses
 - Amount * developPercent goes to development
 - Amount * airDropPercent goes to airDrop
 - Amount * presalePercent goes to presaleWallet
 - Amount * idoPercent goes to idoWallet
 - Amount * partnerPercent goes to partnersWallet
 - Amount * 39.5e18 goes to vest
 - Amount * marketingPercent goes to marketingWallet

Deployer cannot burn or lock user funds

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

Comments:

v1.0

- Deployer can lock claims in Vest for
 - Team
 - Seed
 - Private
- Deployer cannot lock user funds the VersalToken

Deployer cannot pause the contract

Name	Exist	Tested	Verified
Deployer cannot pause	-	-	-



Overall checkup (Smart Contract Security)

Tested	Verified
✓	✓

Legend

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

Modifiers

- VersalToken
 - onlyOwner
 - setVest
 - updateWallet
 - transact
- Vest
 - onlyOwner
 - initialize
 - contractLock
 - isLock
 - claimSeed
 - claimTeam
 - claimPrivate
 - addPrivateWallet

Comments

- While initializing the totalSupply amount of current address is multiplied by privatePercent (17e18) divided by 100e18. The result of this calculation is multiplied by 10/100 and will be send to privateWallet address.
 - Initialize function can be called without any limitations
- ClaimSeed, claimTeam, claimPrivate and addPrivateWallet can be called without any limitations also if there is a isLock modifier because there is a function which can set lockStatus without any limitations by the owner
- claimTeam
 - Can only called if
 - msg.sender is teamWallet
 - teamTime[msg.sender] == 0 or block.timestamp >= teamTime[msg.sender] + 30 days
 - claimCount[msg.sender] < 10
 - Team can only claim 10 times
 - Following amount will send to team address
 - uint amount = totalSupply * teamPercent /100e18;
 - token.transfer(msg.sender,amount*10/100);
- claimSeed
 - Can only called if
 - msg.sender is seedWallet
 - seedTime[msg.sender] == 0 or block.timestamp >= seedTime[msg.sender] + 30 days
 - claimCount[msg.sender] < 10
 - Seed address can only claim 10 times

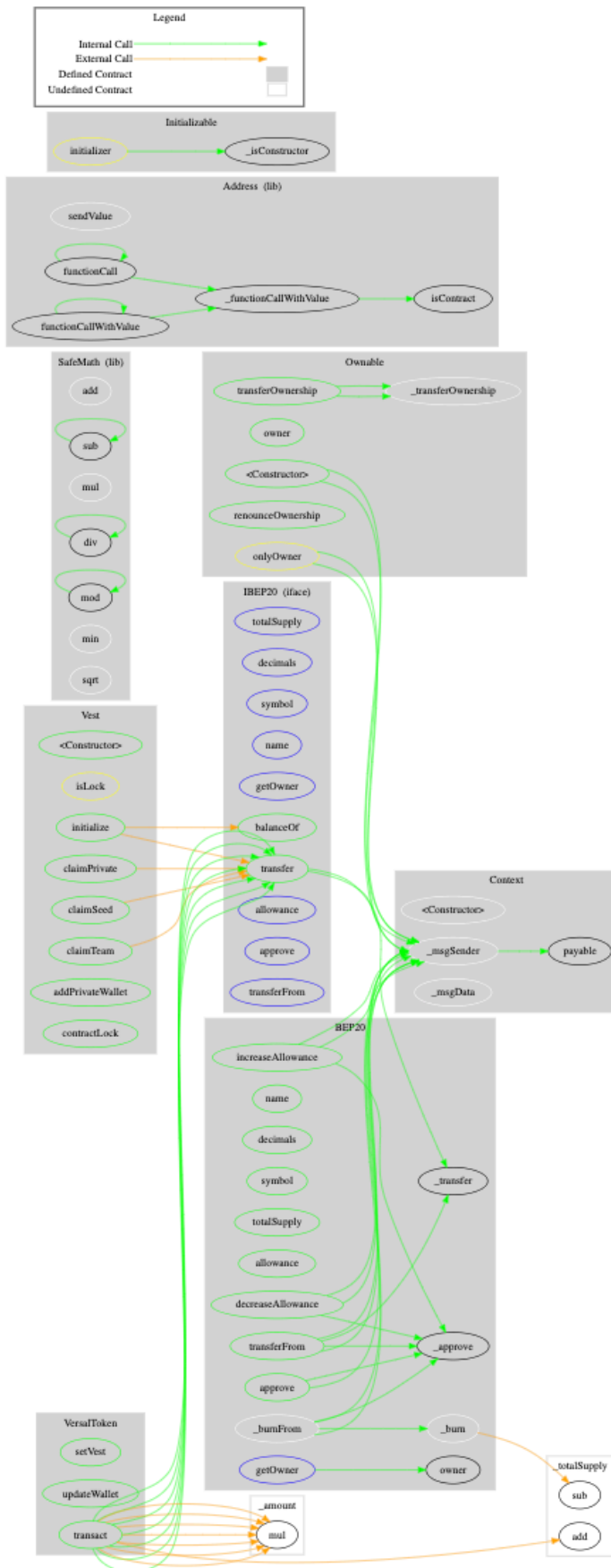
- Following amount will send to seed address
 - `uint amount = totalSupply * seedPercent / 100e18;`
 - `token.transfer(msg.sender, amount * 10 / 100);`

Keep it in mind, if deployer initialize new seed or team address it is possible to claim again 10 times each address

PrivateWallet can add new private details, but cannot be reverted



CallGraph



Source Units in Scope

v1.0

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
	contracts/Vest.sol	3	1	298	219	144	111	113	
	contracts/VersalToken.sol	7	1	928	787	315	484	242	
	Totals	10	2	1226	1006	459	595	355	

v1.1

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
	contracts/Vest.sol	5	1	608	462	260	252	190	
	contracts/VersalToken.sol	7	1	935	794	323	484	257	
	Totals	12	2	1543	1256	583	736	447	

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	All	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

Informational issues

Issue	File	Type	Line	Description
#1	VersalTo ken	State variables that could be declared constant (constable-states)	890, 888, 892, 889, 893, 891,	Add the `constant` attributes to state variables that never change
#2	Vest	State variables that could be declared constant (constable-states)	515, 516 514	Add the `constant` attributes to state variables that never change

Commented Code exist

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

File	Line	Comment
VersalToken	329	// assert(a == b * c + a % b); // There is no case in which this doesn't hold

Recommendation

Remove the commented code, or address them properly.

Audit Comments

15. January 2022:

- Deployer can lock claims in Vest
- Read whole report for more information

16. January 2022:

- Reaudited contracts
 - Issues fixed

SWC Attacks

ID	Title	Relationships	Status
SW C-1 36	Unencrypted Private Data On-Chain	CWE-767: Access to Critical Private Variable via Public Method	PASSED
SW C-1 35	Code With No Effects	CWE-1164: Irrelevant Code	PASSED
SW C-1 34	Message call with hardcoded gas amount	CWE-655: Improper Initialization	PASSED
SW C-1 33	Hash Collisions With Multiple Variable Length Arguments	CWE-294: Authentication Bypass by Capture-replay	PASSED
SW C-1 32	Unexpected Ether balance	CWE-667: Improper Locking	PASSED
SW C-1 31	Presence of unused variables	CWE-1164: Irrelevant Code	PASSED
SW C-1 30	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	PASSED
SW C-1 29	Typographical Error	CWE-480: Use of Incorrect Operator	PASSED
SW C-1 28	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	PASSED

SW C-1 27	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	PASSED
SW C-1 25	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	PASSED
SW C-1 24	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	PASSED
SW C-1 23	Requirement Violation	CWE-573: Improper Following of Specification by Caller	PASSED
SW C-1 22	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	PASSED
SW C-1 21	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	PASSED
SW C-1 20	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	PASSED
SW C-11 9	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards	PASSED
SW C-11 8	Incorrect Constructor Name	CWE-665: Improper Initialization	PASSED
SW C-11 7	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature	PASSED

SW C-11 6	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	PASSED
SW C-11 5	Authorization through tx.origin	CWE-477: Use of Obsolete Function	PASSED
SW C-11 4	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	PASSED
SW C-11 3	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	PASSED
SW C-11 2	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	PASSED
SW C-11 1	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	PASSED
SW C-11 0	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	PASSED
SW C-1 09	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	PASSED
SW C-1 08	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	PASSED
SW C-1 07	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	PASSED
SW C-1 06	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	PASSED

SW C-1 05	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	PASSED
SW C-1 04	Unchecked Call Return Value	CWE-252: Unchecked Return Value	PASSED
SW C-1 03	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	PASSED
SW C-1 02	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	PASSED
SW C-1 01	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	PASSED
SW C-1 00	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	PASSED



*Solidly
Proofed*

Blockchain Security | Smart Contract Audits | KYC


MADE IN GERMANY