SOLIDProof

Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC

MADE IN GERMANY

Audit

Security Assessment

11. January, 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	11. January 2022	• Layout project • Automated- /Manual-Security Testing • Summary

Network

Binance Smart Chain (BEP20)

Website

https://curtisswords.xyz/

Telegram

https://t.me/curtissword

Twitter

http://twitter.com/curtissword_nft

Medium

https://medium.com/@curtissword1

Discord

https://t.co/pbpp1eVNAY

Youtube

https://youtu.be/QYsMzxl15yk

Description

Curtis Sword is a 3D NFT ARPG Metaverse Mobile Game, founded in 2020, with the goal of exploring the value of Tokens and NFT in the encrypted world on blockchain.
Players will have full autonomy to explore and enjoy fun blockchain games, interactive social network space and a wide range of products and services.

Curtis Sword is an innovative blockchain game with more than 100 maps, and our Curtis Sword union is not only a gathering place for game players to communicate, but also a decentralized chemical industry with information integration, venture capital, and union membership autonomy.

The community is similar to the DAO in the encrypted world.Curtis Sword players will be able to purchase and own the mainland identity represented by NFT tokens, collect exclusive NFT collections, and develop and utilize their assets in Curtis Sword(virtual characters, virtual equipment, virtual world items, tools, etc.)

More importantly: The Curtis Sword DAO union will increase the economic value of GameFi through “play to earn”system and distribute game revenue to every union member. In Curtis Sword, union members can use CSC to join the mainland. Our economic system will be fully driven by CSC tokens, used from transactions to the purchase of goods and services, and provide union members the sustainable economic model in a new meta-universe. Our desire is not only to build a game of economic sustainability, but also to create a new type of crypto world where you can have a relaxing moment or get familiar with the financial market, build your own financial system, and learn blockchain's knowledge or follow the union to explore.

The change of the original Facebook name to Meta has surprised the world with the development of "meta universe".Curtis Sword will provide a brand new experience for union members to realize their self-worth in the encrypted world and devote themselves to this new "Meta Universe".

Project Engagement

During the 10th of January 2022, **CurtisSwordCoin 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

CURTIS SWORD

Contract Link

v1.0

- https://bscscan.com/address/0x4eee88962eb0af62c3eb5e8436aa4962995c528f#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.

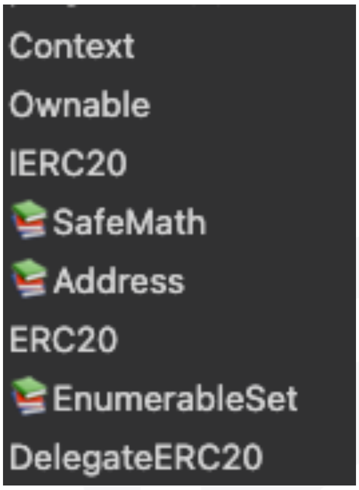
2. Testing and automated analysis that includes the following:
    i) Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
    ii) Symbolic execution, which is analysing a program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



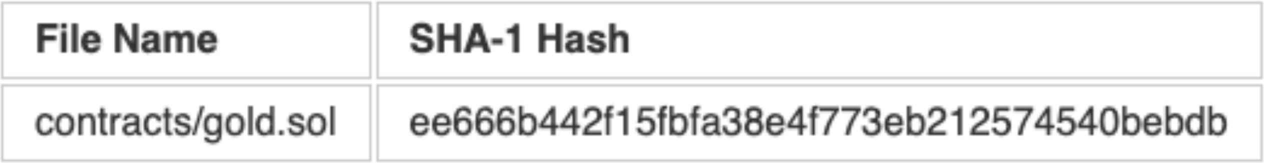
Context
Ownable
IERC20
SafeMath
Address
ERC20
EnumerableSet
DelegateERC20

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

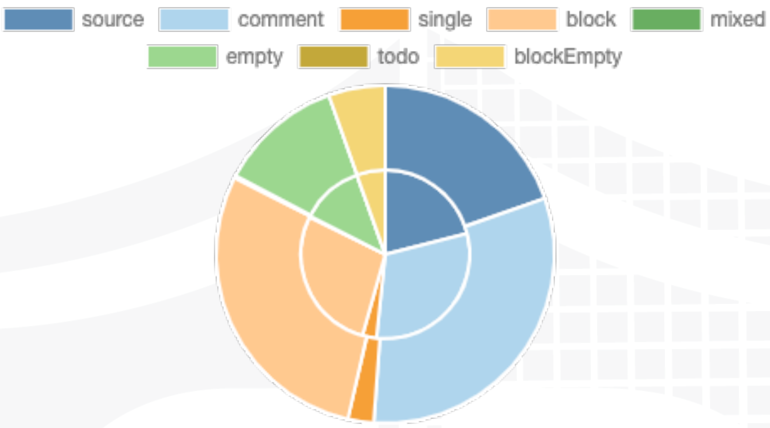


File Name	SHA-1 Hash
contracts/gold.sol	ee666b442f15fbfa38e4f773eb212574540bebdb

Metrics

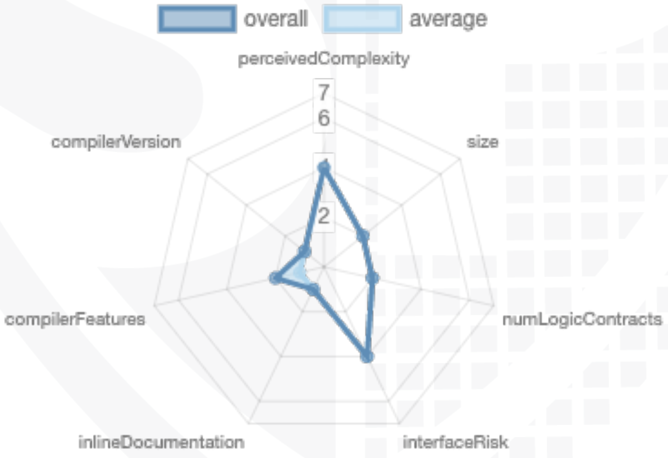
Source Lines

v1.0



Risk Level

v1.0



Capabilities

Components

Version	Contracts	Libraries	Interfaces	Abstract
1.0	3	3	1	2

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	32	0

Version	External	Internal	Private	Pure	View
1.0	10	88	6	10	30

State Variables

Version	Total	Public
1.0	15	5

Capabilities

Version	Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
1.0	`^0.6.9` `^0.6.0`	`ABIEncoderV2`		`yes` (3 asm blocks)	

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

1.0				`yes`	`yes`	

Scope of Work

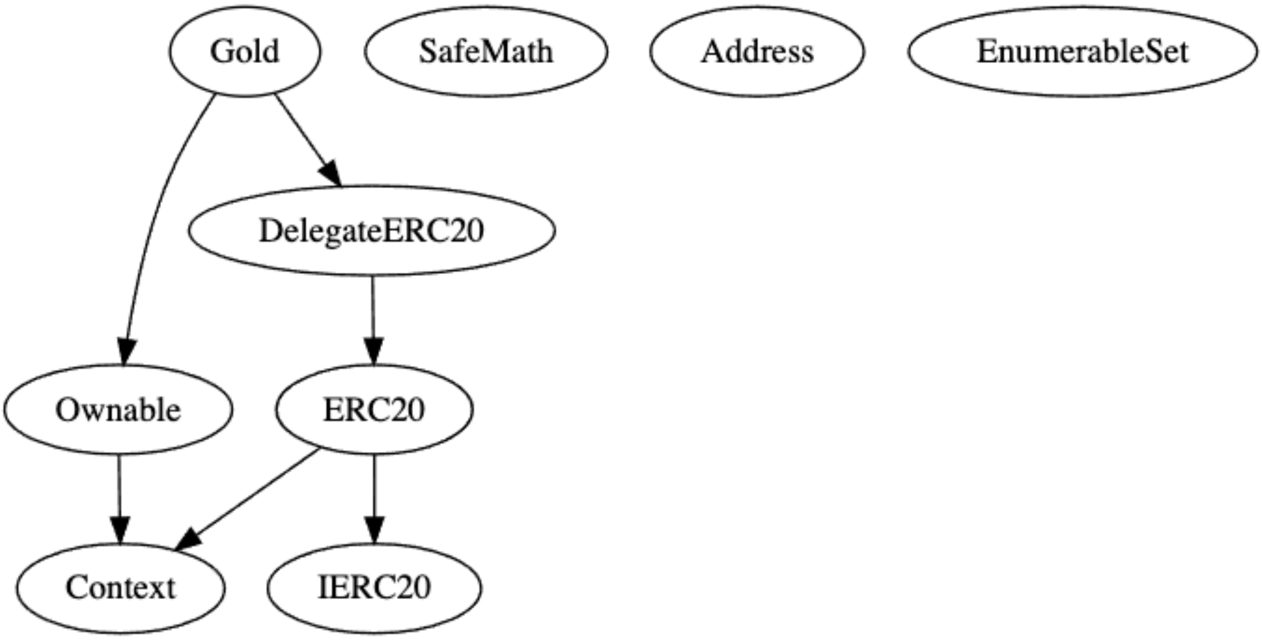
The above token Team provided us with the files that needs to be tested (Github, Bscscan, Etherscan, files, etc.). The scope of the audit is the main contract (usual the same name as team appended with .sol).

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



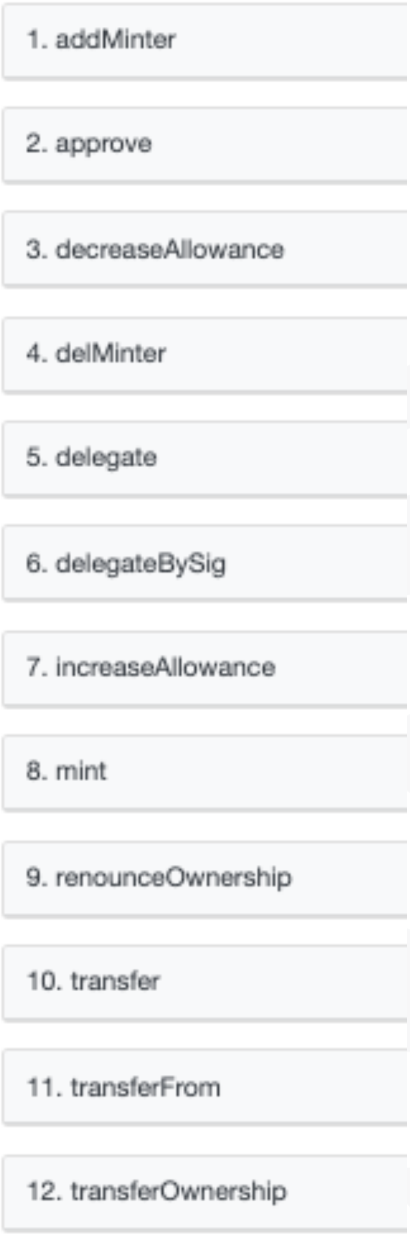
Verify Claims

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



1. addMinter
2. approve
3. decreaseAllowance
4. delMinter
5. delegate
6. delegateBySig
7. increaseAllowance
8. mint
9. renounceOwnership
10. transfer
11. transferFrom
12. transferOwnership

Deployer cannot mint any new tokens

Name	Exist	Tested	Verified
Deployer cannot mint	✓	✓	✗

Max / Total Supply: 100.000.000.000

Comments:

v1.0

- Only Minter can mint
- Only Owner can add/remove new minter
- Minter can only mint if tokens lower than max Supply

Deployer cannot burn or lock user funds

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

Deployer cannot pause the contract

Name	Exist	Tested	Verified
Deployer cannot pause	–	–	–

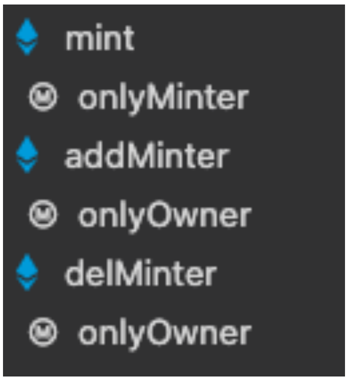
Overall checkup (Smart Contract Security)

Tested	Verified
✓	✓

Legend

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

Modifiers

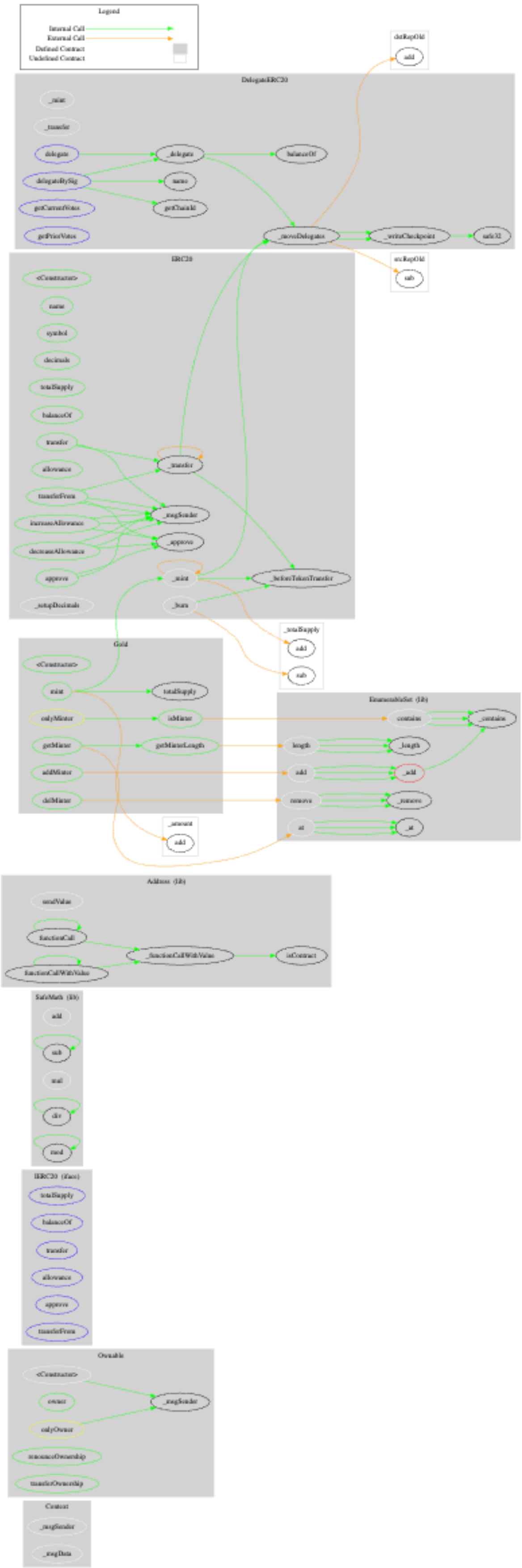


- mint
  - onlyMinter
- addMinter
  - onlyOwner
- delMinter
  - onlyOwner

Comments

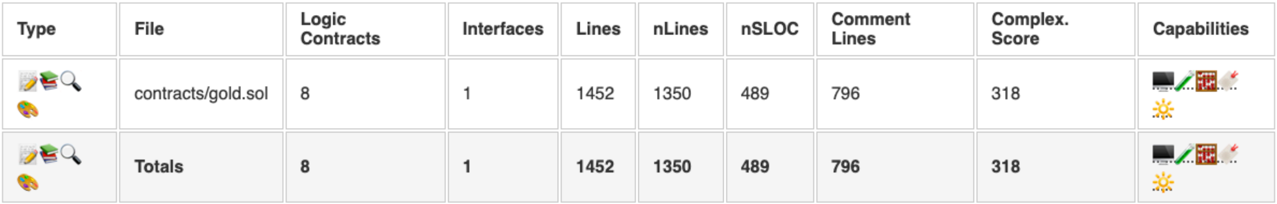
- Deployer can enable/disable following state variables
  - _minters

CallGraph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
dstRepOld
add
DelegateERC20
_mint
_transfer
delegate
_delegate
balanceOf
delegateBySig
name
getCurrentVotes
getChainId
getPriorVotes
_moveDelegates
_writeCheckpoint
safe32
srcRepOld
sub
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
transfer
_transfer
allowance
transferFrom
_msgSender
increaseAllowance
_approve
decreaseAllowance
approve
_mint
_beforeTokenTransfer
_setupDecimals
_burn
_totalSupply
add
sub
Gold
<Constructor>
mint
totalSupply
onlyMinter
isMinter
getMinter
getMinterLength
addMinter
delMinter
_amount
add
EnumerableSet (lib)
contains
_contains
length
_length
add
_add
remove
_remove
at
_at
Address (lib)
sendValue
functionCall
_functionCallWithValue
isContract
functionCallWithValue
SafeMath (lib)
add
sub
mul
div
mod
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Ownable
<Constructor>
owner
_msgSender
onlyOwner
renounceOwnership
transferOwnership
Context
_msgSender
_msgData

Source Units in Scope

v1.0



Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📝📚🔍🎨	contracts/gold.sol	8	1	1452	1350	489	796	318	🖥✅🧮💉🔆
📝📚🔍🎨	**Totals**	**8**	**1**	**1452**	**1350**	**489**	**796**	**318**	🖥✅🧮💉🔆

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	Main	A floating pragma is set	2	The current pragma Solidity directive is „"^0.6.9"".
#3	Main	Local variables shadowing	611	Rename the local variables that shadow another component - Name to name_ - symbol to symbol_

Informational issues

Issue	File	Type	Line	Description

#1	Main	Functions that are not used	520, 477, 488, 504, 515, 412, 450, 27, 834, 879, 1020, 1129, 1058, 1167, 1038, 1146, 1044, 1153, 1030, 1139, 321, 344, 365, 382, 294	Remove unused functions

Commented Code exist

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

Line	Comment
347	// 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

11. January 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	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