SOLIDProof
Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC

MADE IN GERMANY

Metacces

Audit

Security Assessment

12.July.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	9.July,2022	• Layout project • Automated- /Manual-Security Testing • Summary

Network

Binance Smart Chain (BEP20)

Website

www.metacces.com

Description

TBA

Project Engagement

During the 9th of July 2022, Metacces 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 Links

v1.0

- https://bscscan.com/address/0xe52b49b35d10b0c9ebf6398aac4f7cd7a17a43e0#code

- https://bscscan.com/address/0xe9f4556273a117919a02ff90e7ab7aa64481cd07#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.

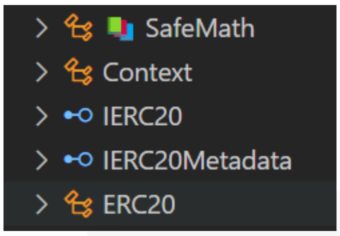
2. Testing and automated analysis that includes the following:
   i) Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii) Symbolic execution, which is analyzing a program to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

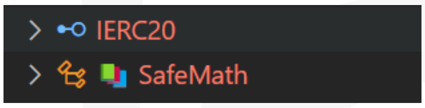
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.

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:


SafeMath
Context
IERC20
IERC20Metadata
ERC20


IERC20
SafeMath

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.

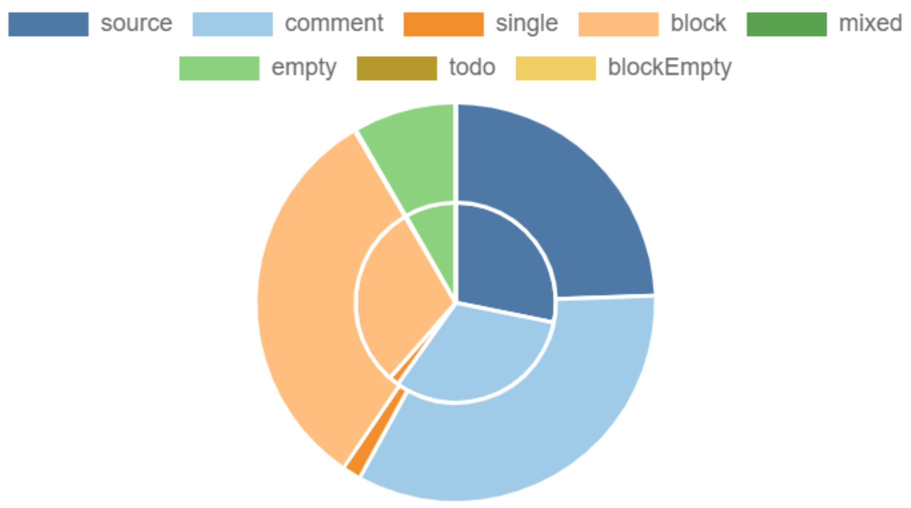
v1.0

File Name	SHA-1 Hash
contracts/BSCBridge.sol	a3e1234de1aefa99dcc643b4b69dcc6ed9e481ca
contracts/Metacces.sol	c8d44b2485ad6d826ec255d1d0a9f1e8febe38b2

Metrics

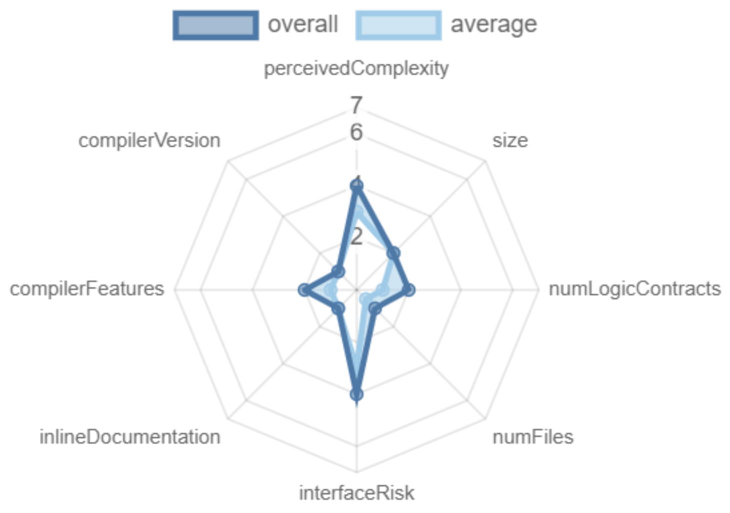
Source Lines

v1.0



Risk Level

v1.0



Capabilities

Components

Version	Contracts	Libraries	Interfaces	Abstract
1.0	3	2	3	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	44	4

Version	External	Internal	Private	Pure	View
1.0	32	64	0	21	18

State Variables

Version	Total	Public
1.0	21	10

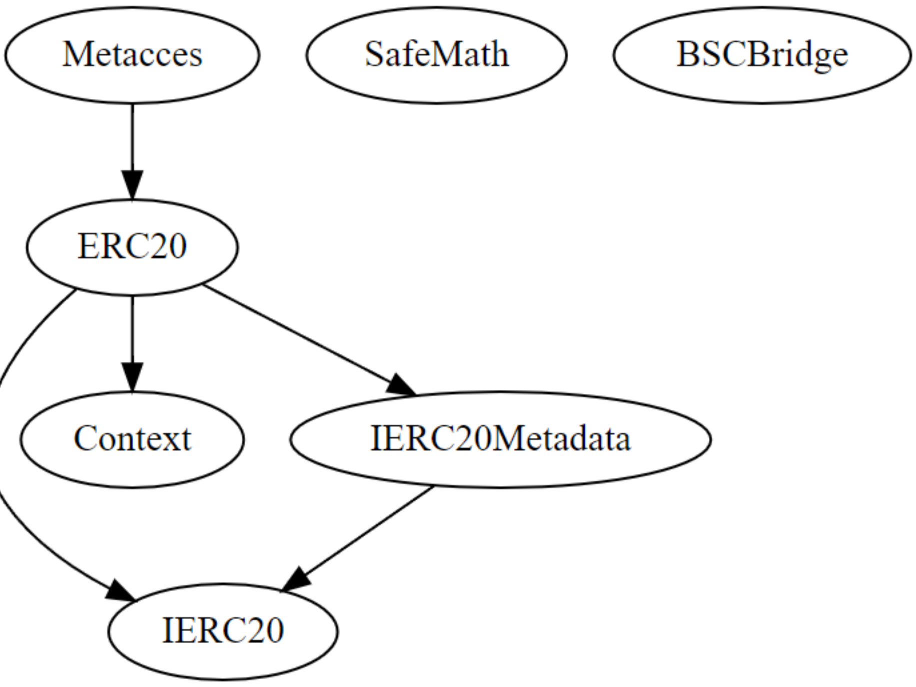
Capabilities

Version	Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
1.0	^0.8.15 ^0.8.0 =0.8.15		Yes		

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

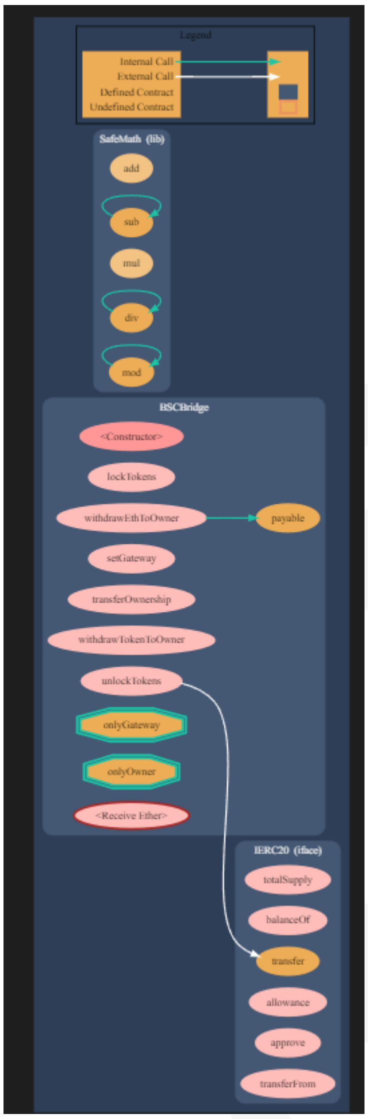
Inheritance Graph

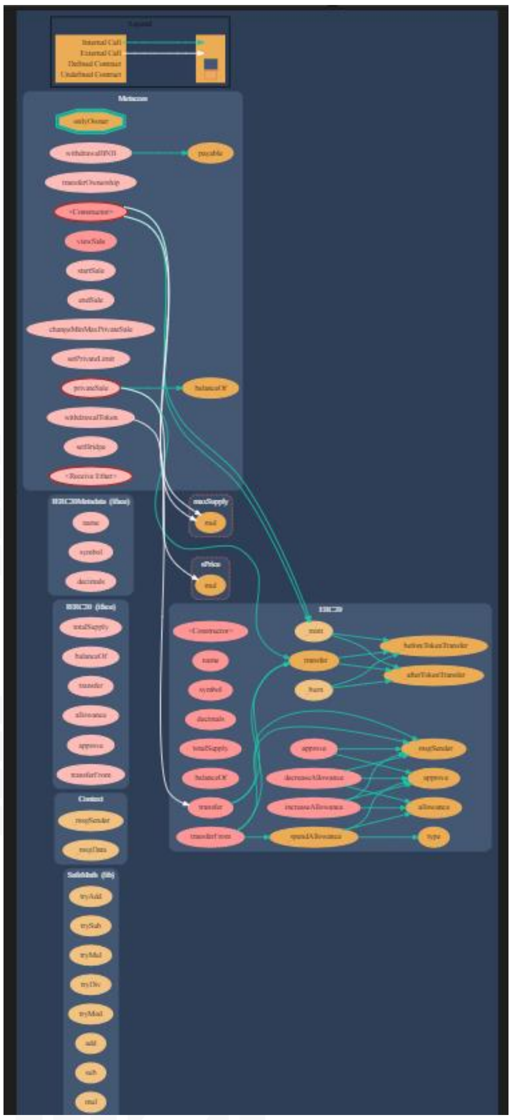
v1.0



Call Graph

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

Is contract an upgradeable

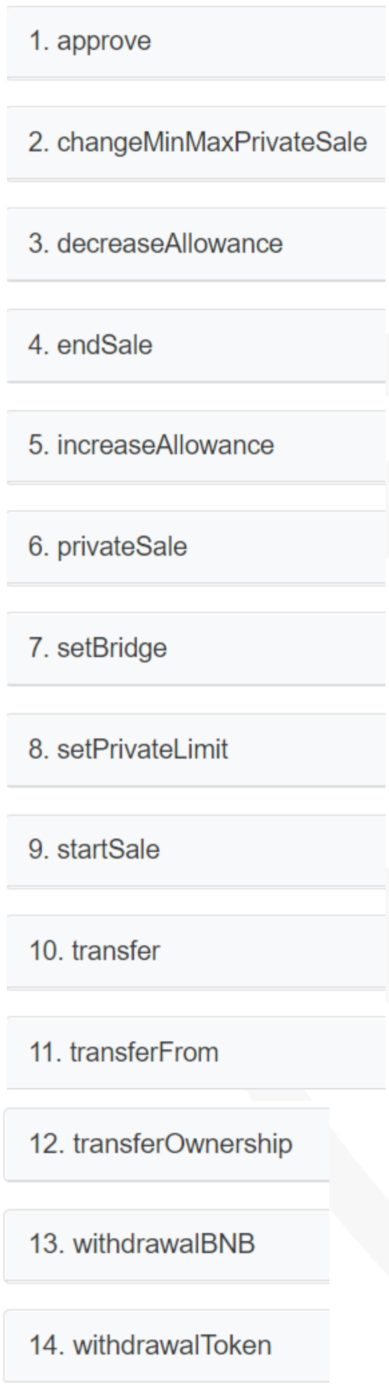
Name	
Is contract an upgradeable?	🟩

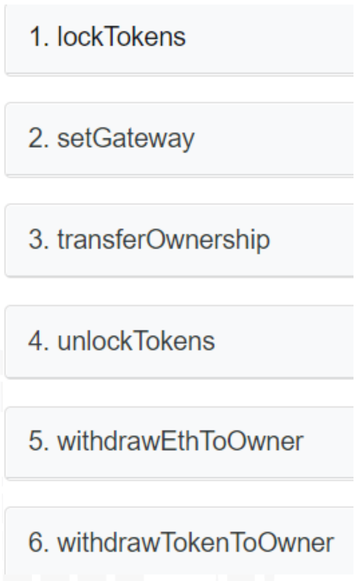
Correct implementation of Token standard

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 contracts

v1.0


1. approve
2. changeMinMaxPrivateSale
3. decreaseAllowance
4. endSale
5. increaseAllowance
6. privateSale
7. setBridge
8. setPrivateLimit
9. startSale
10. transfer
11. transferFrom
12. transferOwnership
13. withdrawalBNB
14. withdrawalToken


1. lockTokens
2. setGateway
3. transferOwnership
4. unlockTokens
5. withdrawEthToOwner
6. withdrawTokenToOwner

Deployer cannot mint any new tokens

Name	Exist	Tested	Status
Deployer cannot mint	🟩	🟩	🟩
Max / Total Supply	500000000		

Deployer cannot burn or lock user funds

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

Deployer cannot pause the contract

Name	Exist	Tested	Status
Deployer cannot pause	🟦	🟦	🟦

Overall checkup (Smart Contract Security)

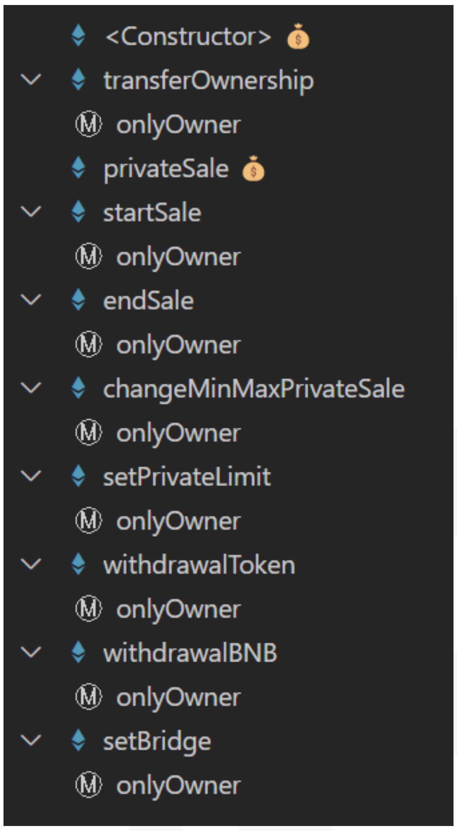
Tested	Verified
🟩	🟩

Legend

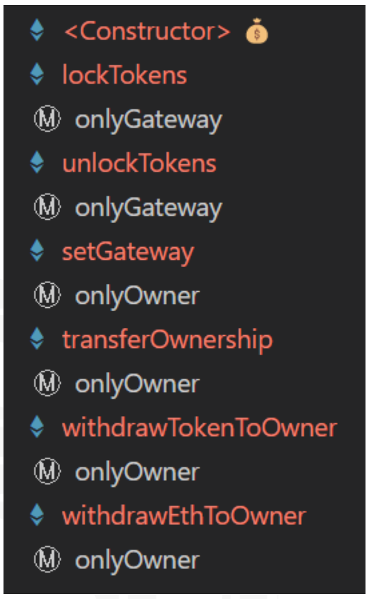
Attribute	Symbol
Verified / Checked	🟩
Partly Verified	🟨
Unverified / Not checked	🟥
Not available	🟦

Modifiers and public functions

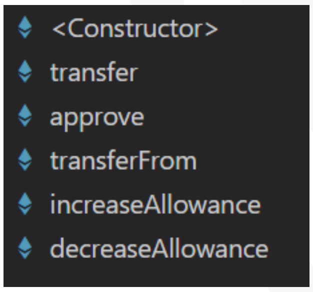
v1.0



```
  <Constructor>
⌄ transferOwnership
    onlyOwner
  privateSale
⌄ startSale
    onlyOwner
⌄ endSale
    onlyOwner
⌄ changeMinMaxPrivateSale
    onlyOwner
⌄ setPrivateLimit
    onlyOwner
⌄ withdrawalToken
    onlyOwner
⌄ withdrawalBNB
    onlyOwner
⌄ setBridge
    onlyOwner
```



```
<Constructor>
lockTokens
  onlyGateway
unlockTokens
  onlyGateway
setGateway
  onlyOwner
transferOwnership
  onlyOwner
withdrawTokenToOwner
  onlyOwner
withdrawEthToOwner
  onlyOwner
```



```
<Constructor>
transfer
approve
transferFrom
increaseAllowance
decreaseAllowance
```

Comments:

- The state variable named "Bridge" serves no purpose in the contract.
- Client's comments:
    - *The owners of the project wants to have their own bridge and they want their users to pay the gas fees from one side and they do it by the gateway from the other side.*
    - *If user A want to swap his tokens from BSC to polygon the following scenario occur:*
      1- User will then click on swap
      2- Then the user send tokens to bsc bridge and the lock event is triggered
      3- on the other side we have a gateway wallet that will transfer the same amount sent to bsc bridge from polygon bridge to the user on the same address since all evm have same address, the gateway wallet pay for the gas fees to transfer the tokens to the user

- *The bridge cannot operate without the Moralis server, so if the Moralis server is down users can work with the tokens normally but they can't swap it between chains*

Source Units in Scope

v1.0

File	**Logic Contracts**	**Interfaces**	**Lines**	**nLines**	**nSLOC**	**Comment Lines**	**Complex. Score**
BSCBridge	2	1	295	240	103	155	71
Metacces	4	2	889	770	316	445	212
Totals	**6**	**3**	**1184**	**1010**	**419**	**600**	**283**

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

Issue	File	Type	Line	Description

#1	Metacces	Withdraw tokens from the contract	859	Owner can withdraw tokens to their address if they want instead of returning it to the users because there is no check to prevent this from happening. Please add a require check that will ensure that the owner can't withdraw the token to their account. Note: Users are advised to be careful while sending tokens because even after this issue is fixed, the owner can still send the tokens to another arbitrary account.

Low issues

Issue	File	Type	Line	Description
#1	All	A floating pragma is set	Top of the file.	The current pragma Solidity directive is „“^0.8.0"".
#2	Metacces.sol	Missing Events	808, 844, 850	Emit an event for critical parameter changes
#3	BSCBridge	Missing Events	259	Emit an event for critical parameter changes

Informational issues

Issue	File	Type	Line	Description
#1	Metacces	Bridge has no functionality	878	There is a function to set the bridge in the contract but there is no functionality or purpose in the contract itself.

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.

12.July,2022:

- Read the whole report and modifiers section 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-1	Hash Collisions With Multiple Variable Length Arguments	CWE-294: Authentication Bypass by Capture-replay	PASSED

33			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-1	Typographical Error	CWE-480: Use of Incorrect Operator	PASSED

29			
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-1	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	PASSED

24			
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-1	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	**PASSED**

20			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-1	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	PASSED

16			
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-1	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	**PASSED**

12			
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-1	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	PASSED

08			
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-1	Unchecked Call Return Value	CWE-252: Unchecked Return Value	**PASSED**

04			
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-1	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	PASSED

0 0			🟩

SolidProof_io @solidproof_io

Solid Proofed

Blockchain Security | Smart Contract Audits | KYC

MADE IN GERMANY