SOLIDProof
Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC Development | Marketing

MADE IN GERMANY

MetaWealth

Audit

Security Assessment

14. October, 2022

For

metawealth

SolidProof_io

@solidproof_io

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	14. October 2022	• Layout project • Automated- /Manual-Security Testing • Summary

Network
Ethereum (ERC20)
Binance Smart Chain (BEP20)

Website
https://www.metawealth.co/

Telegram
https://t.me/metawealthapp

Twitter
https://twitter.com/MetaWealthApp

Facebook
https://www.facebook.com/MetaWealthApp/

Instagram
https://www.instagram.com/metawealthco

LinkedIn
https://www.linkedin.com/company/metawealthapp/

Description
TBA

Project Engagement
During the 13th of October 2022, **MetaWealth 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
- TBA

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:

Dependency / Import Path	Count
@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol	4
@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol	5
@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol	4
@openzeppelin/contracts/access/Ownable.sol	3
@openzeppelin/contracts/interfaces/IERC20.sol	1
@openzeppelin/contracts/token/ERC20/ERC20.sol	2
@openzeppelin/contracts/token/ERC20/IERC20.sol	1
@openzeppelin/contracts/token/ERC721/ERC721.sol	1
@openzeppelin/contracts/token/ERC721/IERC721.sol	2
@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol	1
@openzeppelin/contracts/token/ERC721/extensions/IERC721Metadata.sol	1
@openzeppelin/contracts/utils/Context.sol	2
@openzeppelin/contracts/utils/Strings.sol	1
@openzeppelin/contracts/utils/cryptography/MerkleProof.sol	1
@openzeppelin/contracts/utils/math/SafeMath.sol	1

Tested Contract Files

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

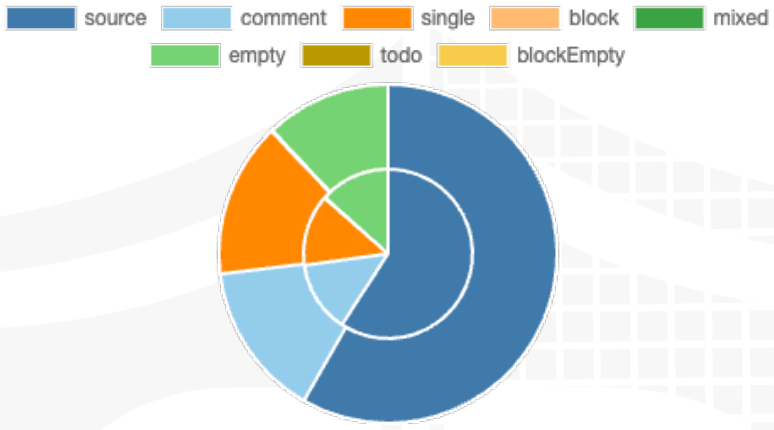
A file with a different Hash has been modified, intentionally or otherwise, after the security review. A different Hash could be (but not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of this review.

v1.0

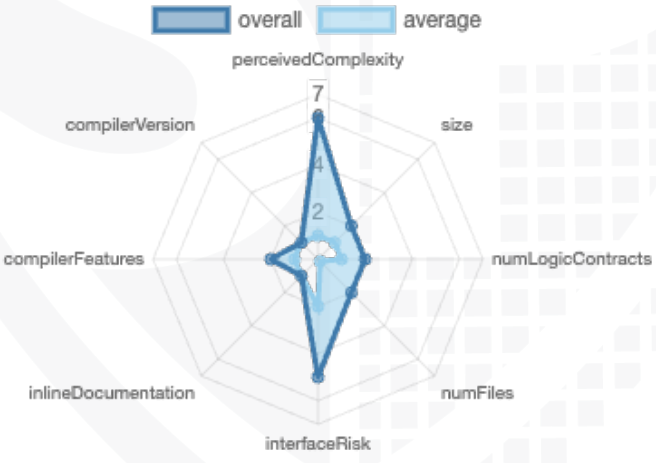
File Name	SHA-1 Hash
contracts/interfaces/IMetaWealthModerator.sol	cac9109b26bca972fc3f80732666f44fe7faf0ed
contracts/interfaces/IAssetVault.sol	4efe7c19aeef1ef484fb53a894d2f5e803ca2e5f
contracts/interfaces/IMetaWealthExchange.sol	36366057a02d793431a2fd93304d08bff8162a30
contracts/interfaces/IMetaWealthAccessControlled.sol	5013f93fa767f739b97db089abbd022181313e59
contracts/interfaces/IMetaWealthFundraiser.sol	7987a29da38871fc1ba1fa86f2d363b04e7072fb
contracts/interfaces/IHeap.sol	fe1c5fd980ec8aef7213864960f555863841f14b
contracts/interfaces/IVaultBuilder.sol	ee13067a8ca9fed0a28ffae59de9f13a43538ebd
contracts/MetaWealthAccessControlled.sol	e6c2c1e0047d3a0868b993b501848ec68f62b94f
contracts/AssetVault.sol	34d8b1c09216be4c601b15f398c4c7130a69f157
contracts/utils/MockERC721.sol	0bd13f438de8a43023fcd98cfdab4b1629fb4e62
contracts/utils/Heap.sol	7af6046d8ca0f9a54ae0d717e1a209a286ef7c50
contracts/utils/MockERC20.sol	2f6ad84de64f9fe44d10c00a2e768195f1315c13
contracts/MetaWealthFundraiser.sol	607191ac560873f15d2355c0464807ef0396e0ce
contracts/MetaWealthModerator.sol	ad212f5cb27748d051b98c46bf81b89d0e62a0db
contracts/MetaWealthExchange.sol	17d06c0d548838948e1d8a376814349ff4883dae
contracts/VaultBuilder.sol	5e8016dba939a77278e9502a75aa31c41daddf6e

Metrics

Source Lines
v1.0



Risk Level
v1.0



Capabilities

Components

Version	Contracts	Libraries	Interfaces	Abstract
1.0	10	0	9	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	75	0

Version	External	Internal	Private	Pure	View
1.0	57	74	0	2	28

State Variables

Version	Total	Public
1.0	25	13

Capabilities

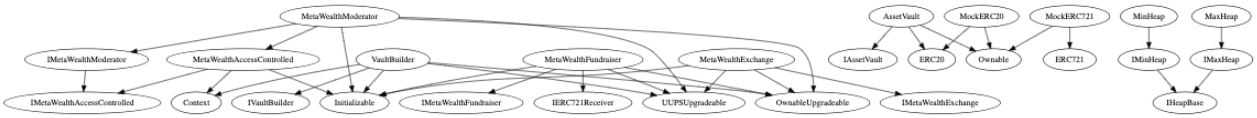
Version	Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
1.0	`^0.8.7` `^0.8.4`				

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

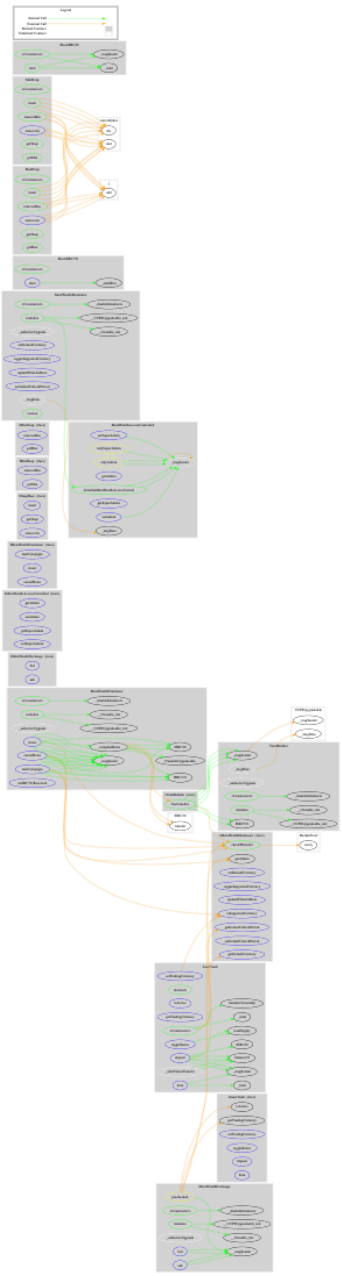
1.0	`yes`			`yes`		`yes` → `NewContract:MinHeap` → `NewContract:MaxHeap` → `NewContract:AssetVault`

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. Is contract an upgradeable
2. Deployer cannot mint any new tokens
3. Deployer cannot burn or lock user funds
4. Deployer cannot pause the contract
5. Deployer cannot set fees
6. Deployer cannot blacklist/antisnipe addresses
7. Overall checkup (Smart Contract Security)

Is contract an upgradeable

Name	
Is contract an upgradeable?	Yes

Comments:

v1.0

- Owner can deploy a new version of the contract which can change any limit and give owner new privileges
    - Be aware of this and do your own research for the contract which is the contract pointing to

Deployer cannot mint any new tokens

Name	Exist	Tested	Status
Deployer cannot mint	✓	✓	✓

Deployer cannot burn or lock user funds

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

Comments:

v1.0

- Tokens
  - can be burned by the owner

Deployer cannot pause the contract

Name	Exist	Tested	Status
Deployer cannot pause	–	–	–

Deployer cannot set fees

Name	Exist	Tested	Status
Deployer cannot set fees over 25%	–	–	–
Deployer cannot set fees to nearly 100% or to 100%	–	–	–

Deployer can blacklist/antisnipe addresses

Name	Exist	Tested	Status
Deployer cannot blacklist/antisnipe addresses	–	–	–

Overall checkup (Smart Contract Security)

Tested	Verified
✓	✓

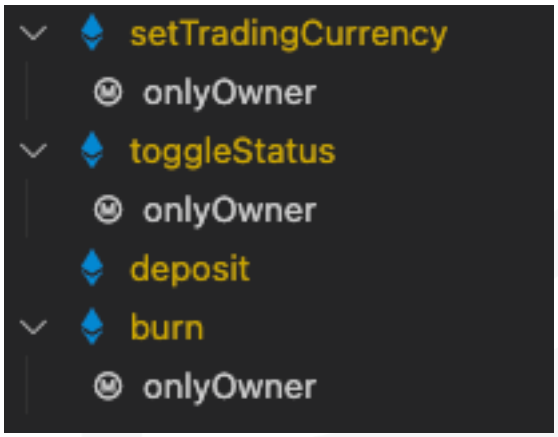
Legend

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

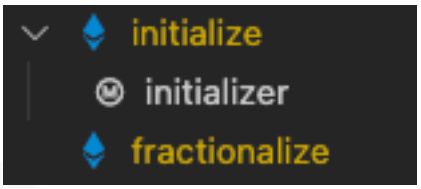
Modifiers and public functions

v1.0

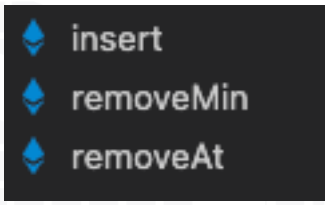
AssetVa



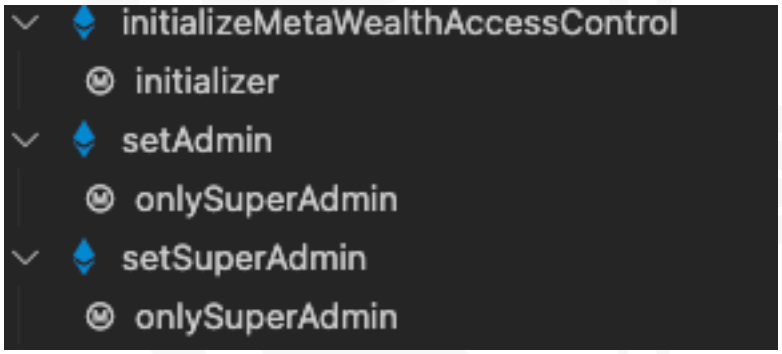
VaultBuilder



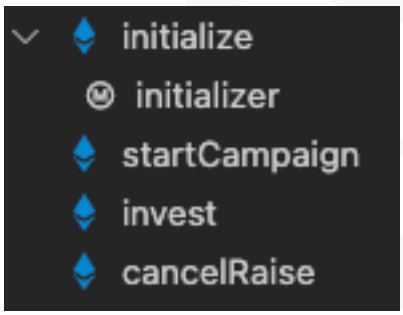
Heap



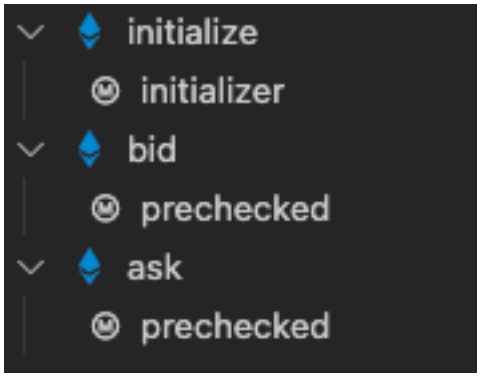
MetaWealthAccessControlled



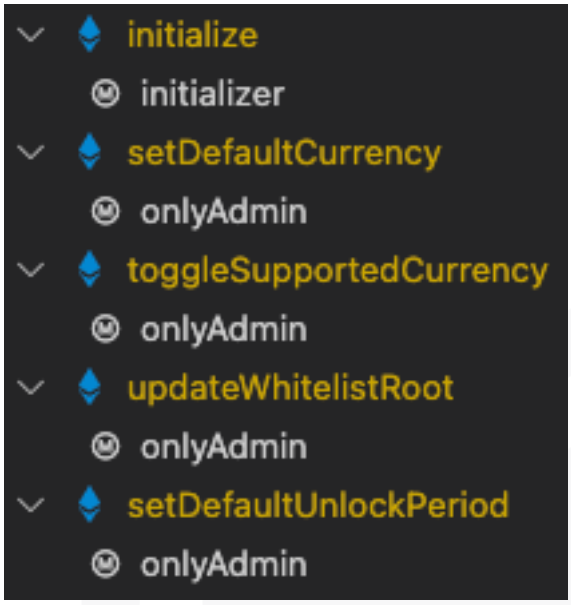
MetaWealthFundraiser



MetaWealthExchange



MetaWealthModerator



Comments

- *Deployer can set following state variables without any limitations*
  - MetaWealthModerator
    - defaultUnlockPeriod
      - Max 2^64 - 1

- *Deployer can enable/disable following state variables*
  - MetaWealthModerator
    - supportedCurrencies
  - AssetVault
    - Active

- *Deployer can set following addresses/string*
  - MetaWealthModerator
    - whitelistRoot
    - defaultCurrency
  - AssetVault
    - tradingCurrency

- *Existing Modifiers*
  - onlySuperAdmin
  - onlyAdmin
  - prechecked

- The owner is able to set some addresses into the merle root which is allowed to call every function where the whitelist is checked.

- There are more than 1 authority in the contracts. Be aware of it because the owner can add whitelisted addresses that are able to call functions where is only restricted to them

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/interfaces/IMetaWealthModerator.sol	———	1	63	28	7	33	19	———
🔍	contracts/interfaces/IAssetVault.sol	———	1	49	24	6	28	13	———
🔍	contracts/interfaces/IMetaWealthExchange.sol	———	1	42	23	5	21	5	———
🔍	contracts/interfaces/IMetaWealthAccessControlled.sol	———	1	31	18	4	18	9	———
🔍	contracts/interfaces/IMetaWealthFundraiser.sol	———	1	82	48	19	37	7	———
🔍	contracts/interfaces/IHeap.sol	———	3	51	23	11	23	21	———
🔍	contracts/interfaces/IVaultBuilder.sol	———	1	34	28	9	17	3	———
📝	contracts/MetaWealthAccessControlled.sol	1	———	65	61	44	7	33	———
📝	contracts/AssetVault.sol	1	———	155	137	97	18	75	———
📝	contracts/utils/MockERC721.sol	1	———	15	15	10	1	10	———
📝	contracts/utils/Heap.sol	2	———	227	227	173	11	102	———
📝	contracts/utils/MockERC20.sol	1	———	15	15	11	1	12	———
📝	contracts/MetaWealthFundraiser.sol	1	———	163	151	112	7	106	📩Σ
📝	contracts/MetaWealthModerator.sol	1	———	143	101	74	9	58	🧱
📝	contracts/MetaWealthExchange.sol	1	———	198	171	149	11	114	🌀♻️
📝	contracts/VaultBuilder.sol	1	———	97	73	55	7	50	🌀
📝🔍	**Totals**	**10**	**9**	**1430**	**1143**	**786**	**249**	**637**	📩🧱🌀♻️Σ

Legend

Attribute	Description
Lines	total lines of the source unit
nLines	normalised lines of the source unit (e.g. normalises functions spanning multiple lines)
nSLOC	normalised 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

Critical issues

No critical issues

High issues

No high issues

Medium issues

Issue	File	Type	Line	Description
#1	MetaWealthFundraiser	Variable will not changed	See description	Because of the contract is using the memory the “CampaignInstance” will not changed in the “invest” function in L84. Make sure to use storage instead or use directly the state variable otherwise the remainingRaise will only be updated for the current call instead of permanently.

Low issues

Issue	File	Type	Line	Description
#1	All	A floating pragma is set	Top of source codes	The current pragma Solidity directive is „“^0.8.7””.
#2	MetaWealthAccessControlled	Missing Zero Address Validation (missing-zero-check)	49, 62	Check that the address is not zero
#3	MetaWealthModerator	Missing Zero Address Validation (missing-zero-check)	43, 77	Check that the address is not zero
#4	AssetVault	State variable visibility is not set	17, 20, 23, 26	It is best practice to set the visibility of state variables explicitly

#5	MetaWealthModerator	State variable visibility is not set	19, 22, 25, 28	It is best practice to set the visibility of state variables explicitly
#6	VaultBuilder	State variable visibility is not set	26	It is best practice to set the visibility of state variables explicitly

Informational issues

Issue	File	Type	Line	Description
#1	Main	NatSpec documentation missing	-	If you started to comment your code, also comment all other functions, variables etc.

Audit Comments

We recommend you to use the special form of comments (NatSpec Format, Follow link for more information https://docs.soliditylang.org/en/latest/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.

14. October 2022:

- Owner can deploy a new version of the contract which can change any limit and give owner new privileges
- Read 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-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	NOT 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

SolidProof_io

@solidproof_io

Blockchain Security | Smart Contract Audits | KYC Development | Marketing

MADE IN GERMANY