SOLIDProof
Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC
Development | Marketing

MADE IN GERMANY

KERC

Audit

Security Assessment
18. July, 2023

For

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 the security or functionality of the technology we agree to analyze.

Version	Date	Description
1.0	26. June 2023	• Layout project • Automated- /Manual-Security Testing • Summary
1.1	28. June 2023	• Reaudit
1.2	18. July 2023	• Updated Token Contract Review

Note - This Audit report comprises a security analysis of the **KERC** smart contracts. This analysis did not include functional testing (or unit testing) of the contract's logic.

Network
Arbitrum One

Website
https://kerc.io

Discord
https://discord.gg/kerc

Description

Krypto ERC (KERC) is the next-gen traditional business, and we provide our users with new opportunities by connecting two major industries: primary and alternative healthcare (PAH) and cryptocurrency. We transform the traditional brick-and-mortar primary care businesses into turnkey tech-driven crypto assets and allow investors of any capital size to access high-yielding sustainable earnings in USDC directly from our investments into the traditional businesses which usually require enormous amounts of capital, knowledge, and compliance to participate.

Project Engagement

During the Date of 23 June 2023, **KERC Team** engaged Solidproof.io to audit smart contracts that they created. The engagement was technical 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

- https://github.com/KERC20/audit-token-and-presale
- Commit: 219d0e956769003112a5dba3c1657b9c7c0405a3

v1.1

- https://github.com/KERC20/audit-token-and-presale
- Commit:89a0070

v1.2

- **Deployed Token Address -** https://arbiscan.io/address/0x2389f6A46562AE5f1557Db8562c39Ef553f3832B
- The remaining contracts are the same as the previous version

Note for Investors: We only Audited a token, presale, and a vesting contract for the **KERC Team**. However, Suppose the project has other contracts (for example, a staking contract etc.) that were not provided to us in the audit scope. In that case, we cannot comment on its security and are not responsible for it in any way.

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.

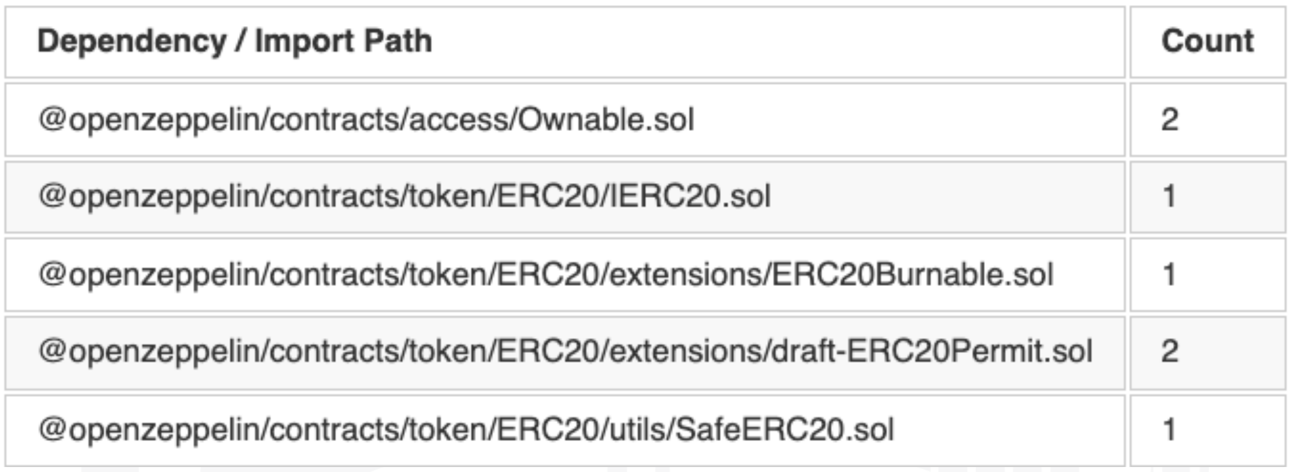
2. Testing and automated analysis that includes the following:
   i) Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii) Symbolic execution, which is analysing a program to 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/access/Ownable.sol	2
@openzeppelin/contracts/token/ERC20/IERC20.sol	1
@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol	1
@openzeppelin/contracts/token/ERC20/extensions/draft-ERC20Permit.sol	2
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.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/tokens/KERC.sol	f887450294c0e06a5531201ad5842839f25f5e86
contracts/vesting/Vesting.sol	82aefd8dff91808f0faaa6ea40670b28db25b8f4
contracts/raise/Presale.sol	ef8e2a74c8c0257ed9e9f3d92e8490a23314ea58

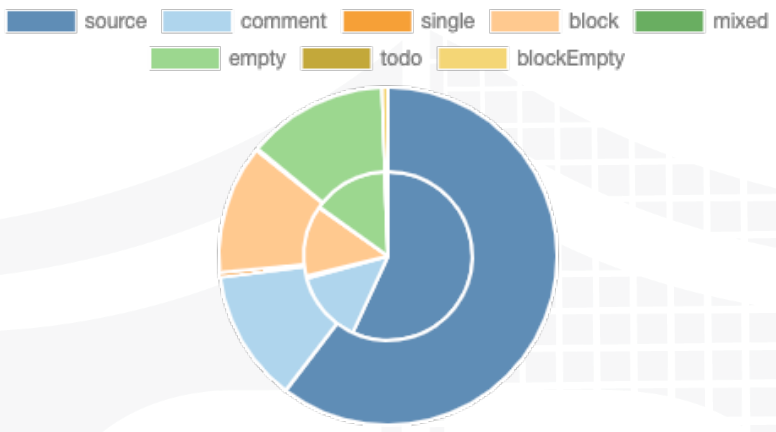
v1.2

File Name	SHA-1 Hash
contracts/tokens/KERC.sol	b4fb5c7bc53b580f78bc196da75202eadf37ac48
contracts/vesting/Vesting.sol	e15d6e18fc171baf90e43c259407e80da8d28df5
contracts/raise/Presale.sol	e0765956a29aa97001c47413715d165ce83bcf4e

Metrics

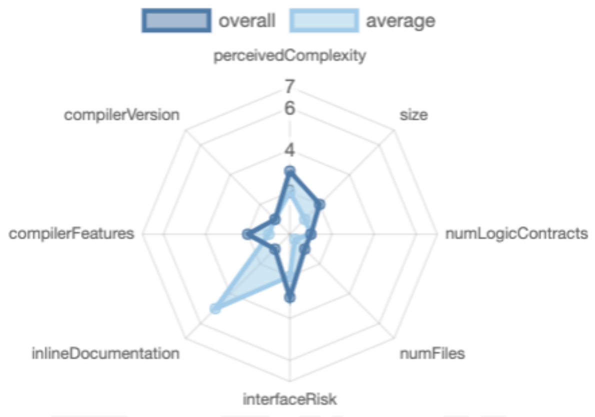
Source Lines

v1.2



Risk Level

v1.2



Capabilities

Components

📝Contracts	📚Libraries	🔍Interfaces	🎨Abstract
3	0	0	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.

🌐Public	💰Payable
18	0

External	Internal	Private	Pure	View
11	17	3	0	7

StateVariables

Total	🌐Public
19	17

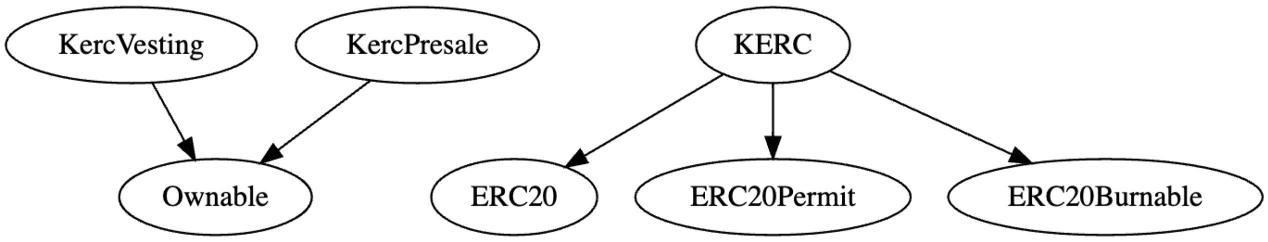
Capabilities

Solidity Versions observed	🧪 Experimental Features	💰 Can Receive Funds	🖥 Uses Assembly	💣 Has Destroyable Contracts
`=0.8.18`				

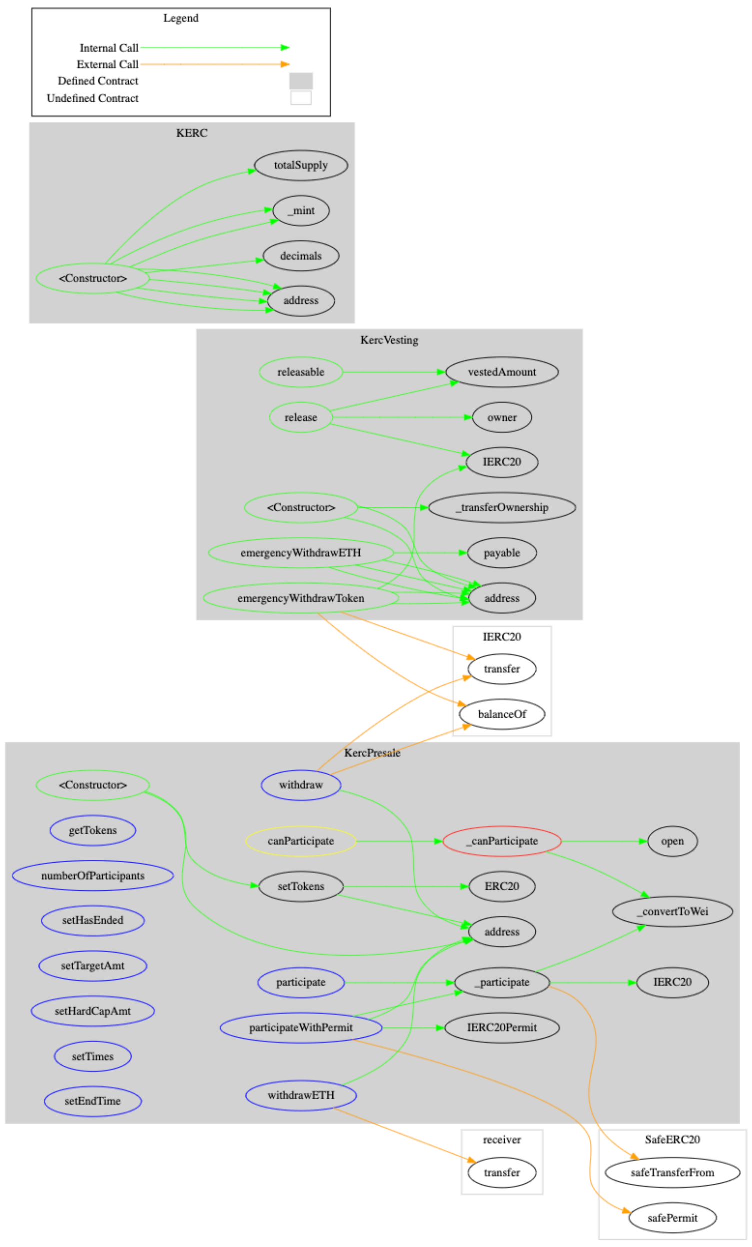
📤 Transfers ETH	⚡ Low-Level Calls	👥 DelegateCall	🧮 Uses Hash Functions	🔖 ECRecover	🌀 New/Create/Create2
`yes`					`yes` → `NewContract:KercVesting`

♻️ TryCatch	Σ Unchecked
	`yes`

Inheritance Graph
v1.2



CallGraph
v1.2



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. Deployer cannot set fees
7. Deployer cannot blacklist/antisnipe addresses
8. Overall checkup (Smart Contract Security)

Is contract an upgradeable

Name	
Is contract an upgradeable?	No

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	✓	✓	✓

Deployer cannot mint any new tokens

Name	Exist	Tested	Status
Deployer cannot mint	✓	✓	✓
Max / Total Supply	500_000_000		

Comments:

v1.2

- Owner cannot mint new tokens manually

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	✓	✓	✓

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)

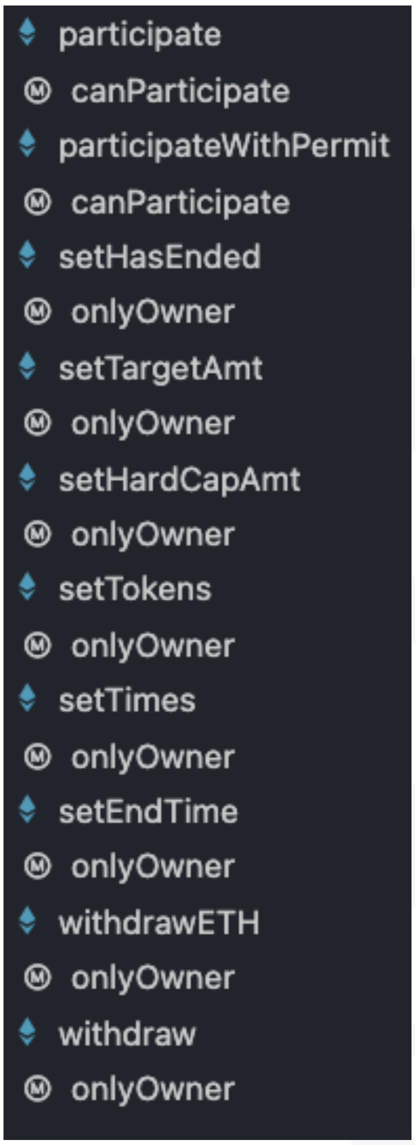
Tested	Verified
✓	✓

Legend

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

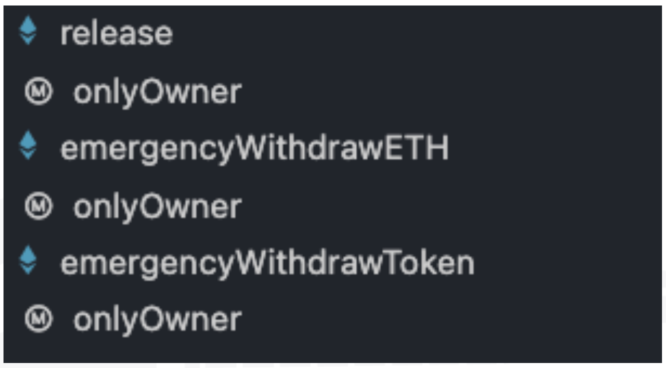
Modifiers and public functions
v1.2

Presale



```
participate
canParticipate
participateWithPermit
canParticipate
setHasEnded
onlyOwner
setTargetAmt
onlyOwner
setHardCapAmt
onlyOwner
setTokens
onlyOwner
setTimes
onlyOwner
setEndTime
onlyOwner
withdrawETH
onlyOwner
withdraw
onlyOwner
```

Vesting



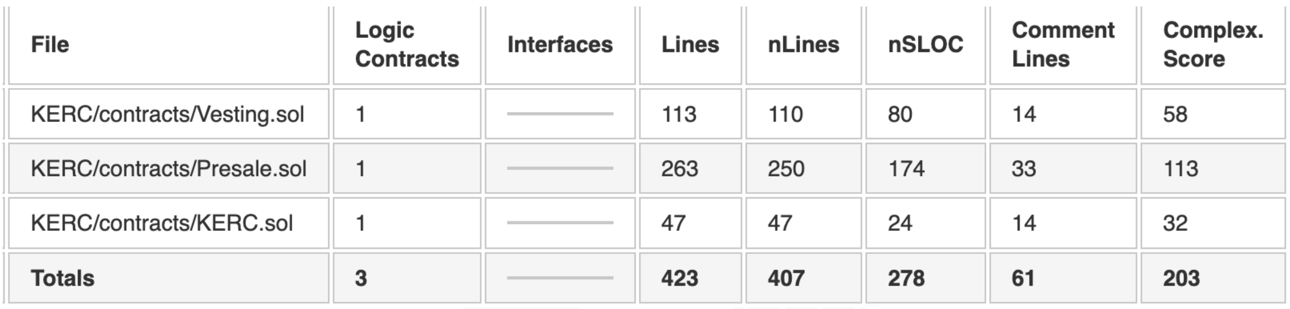
```
release
onlyOwner
emergencyWithdrawETH
onlyOwner
emergencyWithdrawToken
onlyOwner
```

Ownership Privileges

- Presale.sol -
  - Set/Update the end time, but the new end time must be greater than the existing end time, which means the sale can only be extended.
  - Set/Update the start and end time of the deposit
  - Set supported tokens for the presale
  - Withdraw any tokens from the contract balance to the "raiseReceiver" address.
  - The owner can change the hard cap of the presale at anytime

- *Vesting.sol* -
  - Withdraw ETH and foreign tokens from the contract. The owner cannot withdraw vested tokens from the contract. They will be released according to the vesting schedule.
  - Only the owner can release vested tokens to their own wallet.

Source Units in Scope
v1.2



File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score
KERC/contracts/Vesting.sol	1	—	113	110	80	14	58
KERC/contracts/Presale.sol	1	—	263	250	174	33	113
KERC/contracts/KERC.sol	1	—	47	47	24	14	32
Totals	**3**	—	**423**	**407**	**278**	**61**	**203**

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

No medium issues

Low issues

Issue	File	Type	Line	Description	Status
#1	Vesting.sol	Missing Zero Address Validation (missing-zero-check)	33	Check that the address is not zero in the constructor because it cannot be changed once the contract is deployed.	Fixed
#2	Presale.sol	Missing Events Arithmetic	174 — 188	Emit an event for critical parameter changes	Fixed

Informational issues

Issue	File	Type	Line	Description	Status
#1	All	NatSpec documentation missing	—	If you started to comment your code, also comment all other functions, variables etc.	Open
#2	Presale.sol	Missing zero value check	60	Check that the value is not zero	Fixed

Audit Comments

We recommend you use the particular form of comments (NatSpec Format, Follow the 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 variable, functions etc., do.

18. July 2023:

- There is still an owner because the contracts are yet to be deployed (The owner still has not renounced ownership)
- 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-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	**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

Solid Proofed

Blockchain Security | Smart Contract Audits | KYC
Development | Marketing

MADE IN GERMANY