SOLIDProof

Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC Development | Marketing

MADE IN GERMANY

Polkastream

Audit

Security Assessment

27. February, 2023

For

$PSTR

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	07. January 2023	• Layout project • Automated- /Manual-Security Testing • Summary
	16. January 2023	• Report adjustments
	02. February 2023	• Report adjustments
1.1	16. February 2023	• Reaudit

Network
Binance Smart Chain (BEP20)

GitHub
https://github.com/polkastream-studios

Website
https://polkastream.io

Medium
medium.com/polkastream

Telegram
https://t.me/polkastream

Discord
https://discord.gg/polkastream

Twitter
https://twitter.com/polkastream

Reddit
https://www.reddit.com/user/polkastream

Description

Polkastream is poised to be the next innovative addition to the Polkadot parachain system by building an integrated and intelligently monetized platform for the Web 3.0 streaming and gaming market. On the Polkastream platform, users can listen to music and podcasts, watch movies and TV shows, as well as socialize and play over 40 games in the Polkaverse. In return, streamers and gamers are rewarded with the $PSTR token, which is uniquely designed with incentivization and token-burning strategies for better rewards, long-term crypto market stability, and data security for all digital media users.

Project Engagement

On January 7, 2023, the **Polkastream.io Team** contacted Solidproof.io to audit the **$PSTR** smart contract that they developed. The engagement was technical in nature and focused on identifying security flaws in the design and implementation of the smart contract. On January 12, the **Polkastream.io Team** further requested to audit the **$PSTR** tokenomics. They provided Solidproof.io with access to the $PSTR GitHub code repository at https://github.com/polkastream/pstr and whitepaper.

Logo

Contract Link

v1.0

- Github
  - https://github.com/polkastream/pstr/blob/main/contracts/contract.sol
  - Commit
    - https://github.com/polkastream/pstr/commit/f0a5b9a2523bb43da1c026d67804da4b55fcc755
- https://bscscan.com/address/0x3cdd71d99cb393928b74d549d4cb0a6ffe0a60a8#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, the contract was 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 if the code does what it is supposed to according to the specifications, sources, and instructions that were provided to Solidproof.

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:

```
IERC20
Ownable
```

Tested Contract Files

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

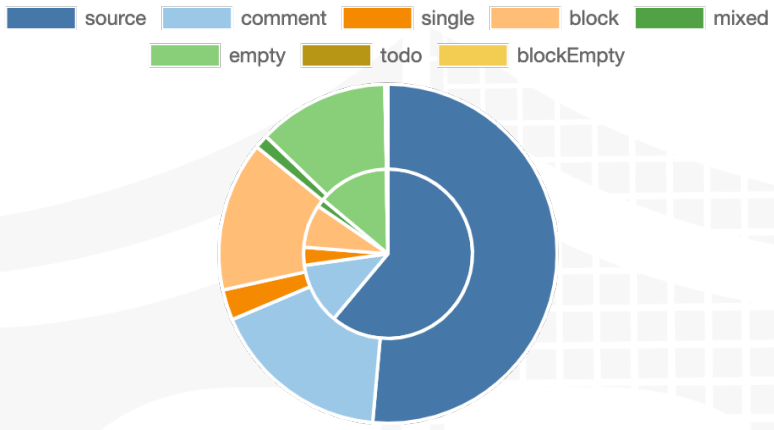
A file with a different Hash has been modified, intentionally or otherwise, after the security review. A different Hash could be (but not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of this review.

v1.0

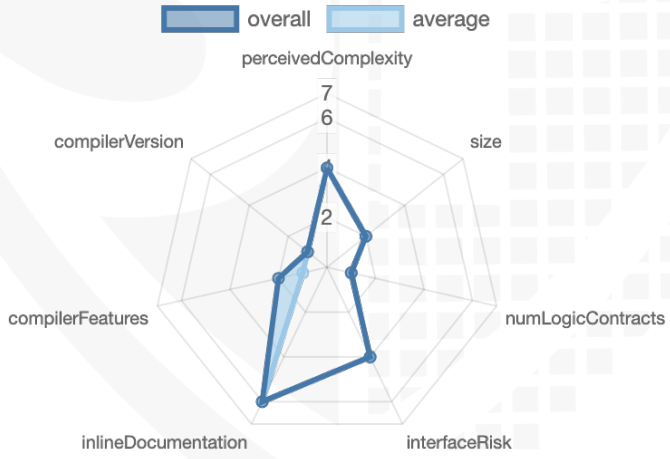
File Name	SHA-1 Hash
contracts/contract.sol	3c03135a94ff9b369c35f36069fe2f3d36b58644

Metrics

Source Lines
v1.0



Risk Level
v1.0



Capabilities

Components

Version	Contracts	Libraries	Interfaces	Abstract
1.0	1	0	1	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	37	0

Version	External	Internal	Private	Pure	View
1.0	10	34	17	5	17

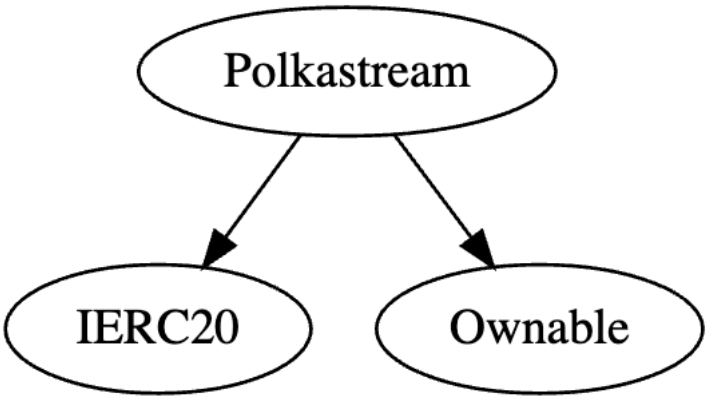
State Variables

Version	Total	Public
1.0	29	13

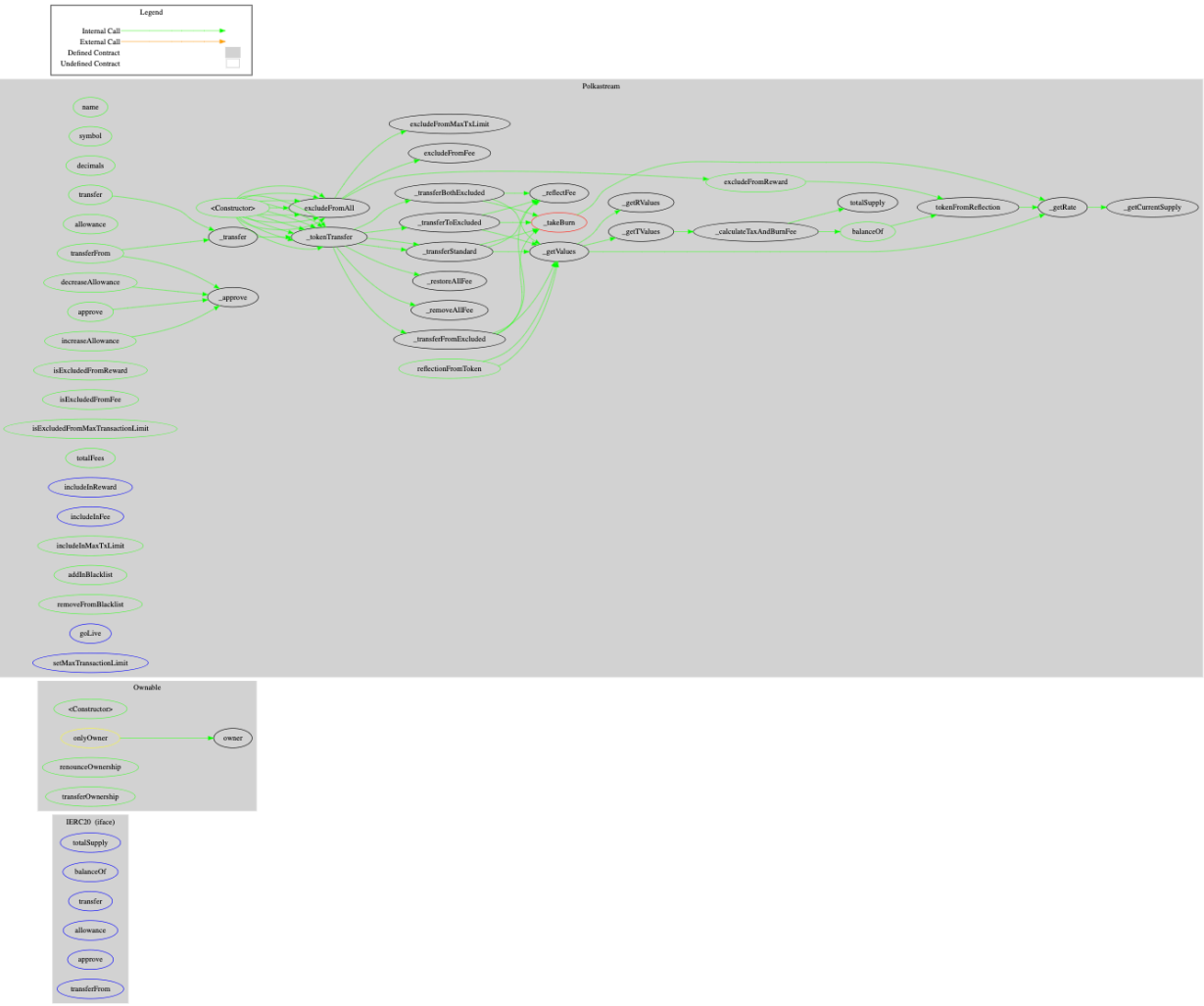
Capabilities

Version	Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
1.0	^0.8.17				

Inheritance Graph
v1.0



CallGraph
v1.0



Scope of Work/Verify Claims

The Polkastream Team provided us with the $PSTR smart contract files via its GitHub repository at https://github.com/polkastream/pstr. The scope of the audit is the main contract (usually the same name as the team appended with .sol).

We will verify the following claims:

1. Is the contract 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 the contract upgradeable

Name	
Is the contract 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	✓	✓	✓

Write functions of contract v1.0

transfer
approve
transferFrom
increaseAllowance
decreaseAllowance
excludeFromReward
includeInReward
excludeFromFee
includeInFee
excludeFromMaxTxLimit
includeInMaxTxLimit
excludeFromAll
addInBlacklist
removeFromBlacklist
goLive
setMaxTransactionLimit

Deployer cannot mint any new tokens

Name	Exist	Tested	Status
Deployer cannot mint	-	-	-
Max / Total Supply			1000000000

Deployer cannot burn or lock user funds

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

Comments:

v1.0

- Owner can lock user funds by
  - blacklisting addresses
  - Setting max tx amount to 0

- 4% fee is applied per transaction. 3% of that fee is distributed to holders, and 1%, up to 50% of the total supply, is permanently burned.

Deployer cannot pause the contract

Name	Exist	Tested	Status
Deployer cannot pause the contract	–	–	–

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	✓	✓	✗

Comments:

v1.0

- Owner is able to blacklist addresses

Overall checkup (Smart Contract Security)

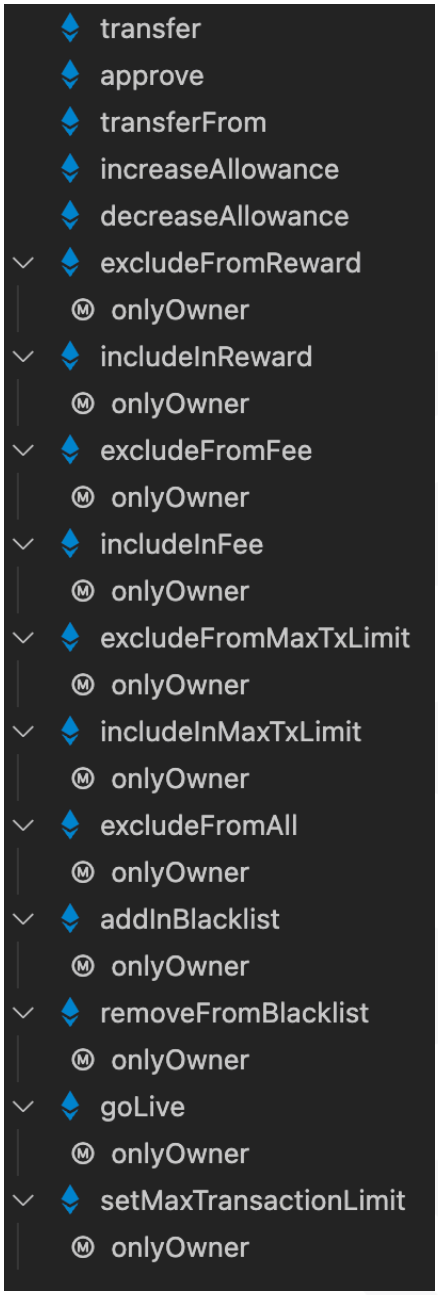
Tested	Verified
✓	✓

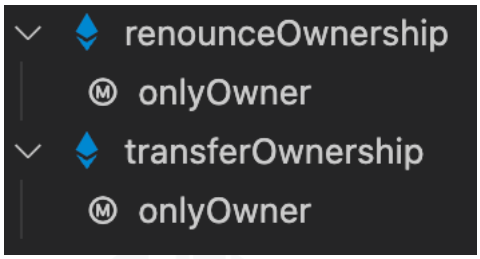
Legend

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

Modifiers and public functions

v1.0





Comments

- *Deployer can set following state variables without any limitations*
  - _maxTxLimit

- *Deployer can enable/disable following state variables*
  - _isBlacklisted
  - _isExcludedFromMaxTxLimit
  - _isExcludedFromFee
  - _isExcluded
  - _excluded

- *Existing Modifiers*
  - onlyOwner

- Addresses that are excluded from fee can buy tokens before token goes live
- Sniper duration can be set without any limitation while going live with the token

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/contract.sol	2	1	546	491	355	118	260	🔆
📝🔍🌈	**Totals**	**2**	**1**	**546**	**491**	**355**	**118**	**260**	🔆

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
#1	Main	Missing Zero Address Validation (missing-zero-check)	228	Check that the address is not zero

Informational issues

No informational issues

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 those variables, functions etc. do.

27. Februrary 2023:

- Please read the entire report and modifiers section for more information.

Testing Results

Polkastream contract

Deployment

✔ Should set the right owner
✔ Should set the right name
✔ Should set the right symbol
✔ Should mint the total supply
✔ Should assign 0 tokens to the owner
✔ Should correctly distribute the total supply among wallets
✔ Should exclude reserved wallets from rewards
✔ Should exclude reserved wallets from fees

Transactions

✔ Should prevent spend before going live
✔ Should prevent spend greater than max tx limit

Blacklist

✔ Should prevent spends from Blacklisted wallets
✔ Should allow spends from Non-Blacklisted wallets
✔ Should blacklist buys close to going live
✔ Should NOT blacklist buys NOT close to going live

Claims

✔ Is the 4% per $PSTR transaction fee calculated correctly?
- Yes

✔ Is 75% of the per transaction fee distributed to all holders?
- Yes

✔ Is 25% of the per transaction fee, up to 50% of the max $PSTR supply, permanently burned?
- Yes

✔ After 50% of the max supply is burned, will the per transaction fee revert to 4% with 100% of it distributed to all holders?
- Yes

✔ Are all company wallets excluded from receiving or paying the per transaction fee before/after the 50% supply burn?
- Yes

✔ Can owner modify the excluded wallets from the per transaction fee?
- Yes

✔ Does $PSTR have a per transaction limit and if so, how many $PSTR is the limit?
- Yes, the initial limit is one million $PSTR

✔ Can the per transaction limit be changed by owner?
- Yes, to the full amount of smart contract

✔ Who can change the $PSTR smart contract owner?
- The owner itself.

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

Blockchain Security | Smart Contract Audits | KYC Development | Marketing

MADE IN GERMANY