SOLIDProof

Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC Development | Marketing

MADE IN GERMANY

Sugar Yield

Audit

Security Assessment

21. January, 2023

For

SUGAR YIELD

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	19. January 2023	• Layout project • Automated- /Manual-Security Testing • Summary

Network
Binance Smart Chain (BEP20)

Website
https://sugaryield.com/

Telegram
https://t.me/Sugaryield

Twitter
https://twitter.com/SugarYield

Discord
https://discord.gg/fKq9Ak7H

Description

TBA

Project Engagement

During the 19th of January 2023, **Sugar Yield 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

- https://testnet.bscscan.com/address/0xf846a2b1a594Bf3D6096Da2b9AE63c220eEAC846#code

- https://bscscan.com/address/0xEea2656A7749BF437f65ad9622dB64C05088Acf8#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.

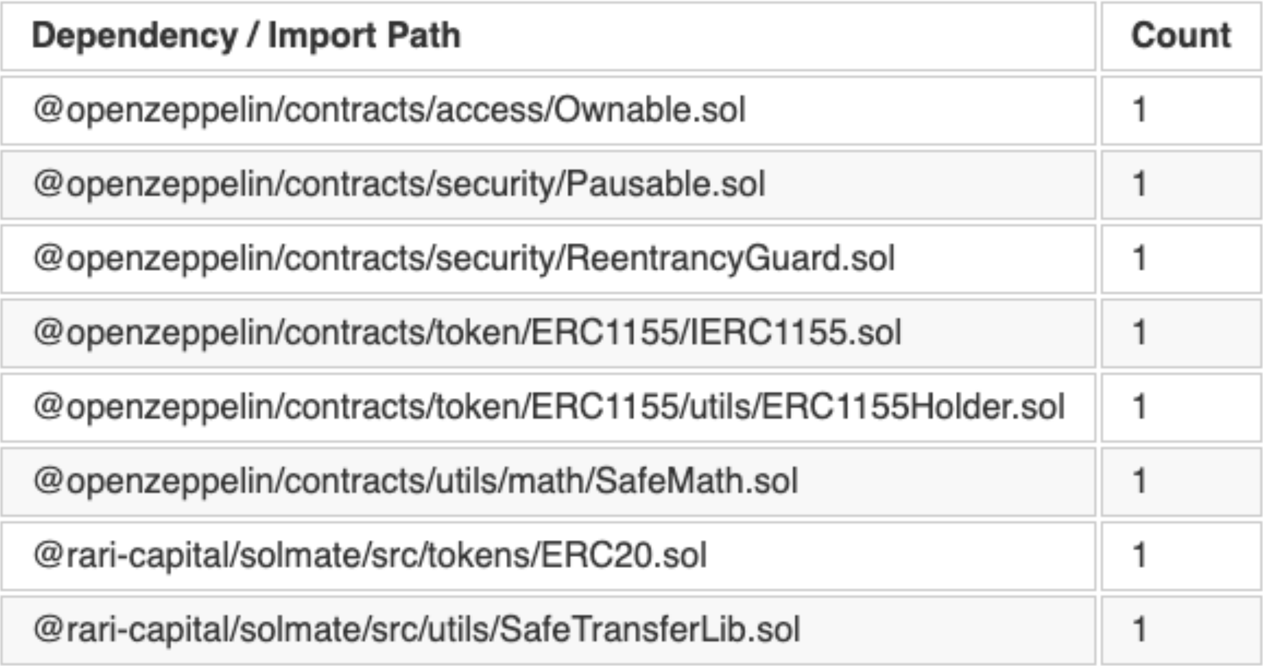
2. Testing and automated analysis that includes the following:
   i) Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii) Symbolic execution, which is analysing a program to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



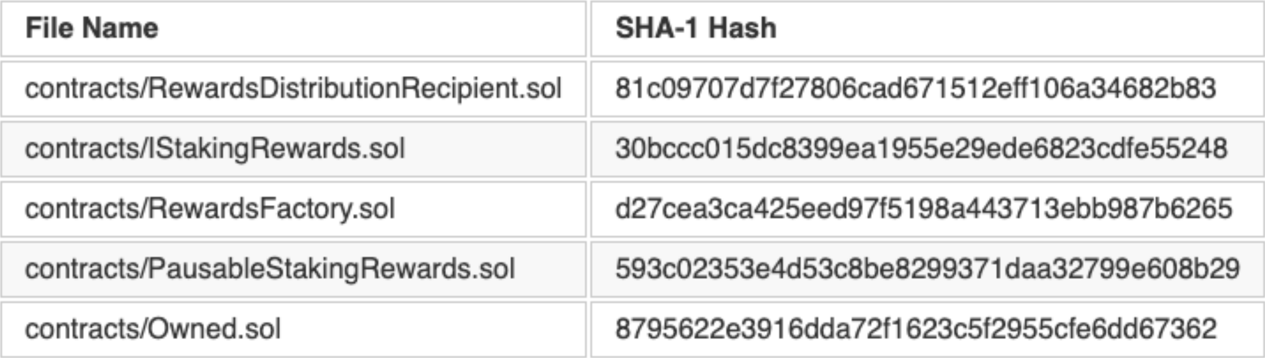
Dependency / Import Path	Count
@openzeppelin/contracts/access/Ownable.sol	1
@openzeppelin/contracts/security/Pausable.sol	1
@openzeppelin/contracts/security/ReentrancyGuard.sol	1
@openzeppelin/contracts/token/ERC1155/IERC1155.sol	1
@openzeppelin/contracts/token/ERC1155/utils/ERC1155Holder.sol	1
@openzeppelin/contracts/utils/math/SafeMath.sol	1
@rari-capital/solmate/src/tokens/ERC20.sol	1
@rari-capital/solmate/src/utils/SafeTransferLib.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

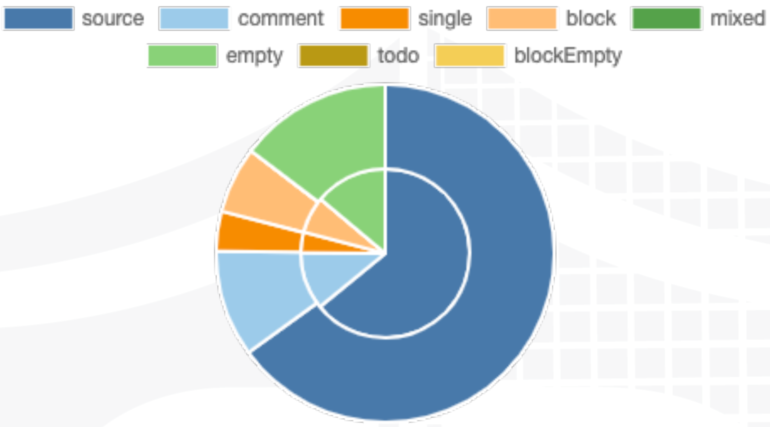


File Name	SHA-1 Hash
contracts/RewardsDistributionRecipient.sol	81c09707d7f27806cad671512eff106a34682b83
contracts/IStakingRewards.sol	30bccc015dc8399ea1955e29ede6823cdfe55248
contracts/RewardsFactory.sol	d27cea3ca425eed97f5198a443713ebb987b6265
contracts/PausableStakingRewards.sol	593c02353e4d53c8be8299371daa32799e608b29
contracts/Owned.sol	8795622e3916dda72f1623c5f2955cfe6dd67362

Metrics

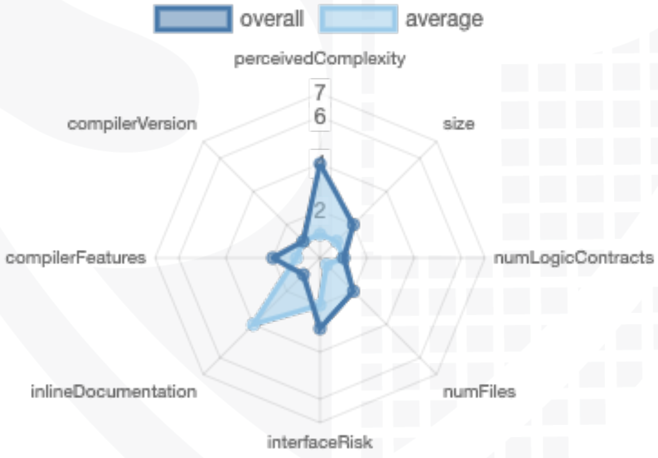
Source Lines

v1.0



Risk Level

v1.0



Capabilities

Components

Version	Contracts	Libraries	Interfaces	Abstract
1.0	3	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	30	0

Version	External	Internal	Private	Pure	View
1.0	25	21	1	0	13

State Variables

Version	Total	Public
1.0	17	15

Capabilities

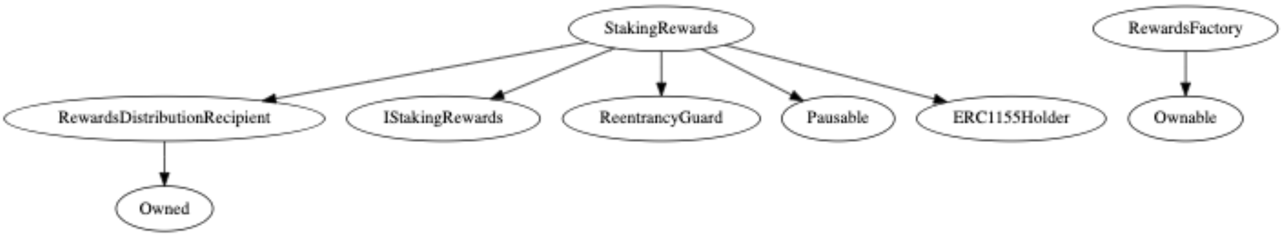
Version	Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
1.0	0.8.15				

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

1.0	`yes`		`yes` → `NewContract:StakingRewards`			

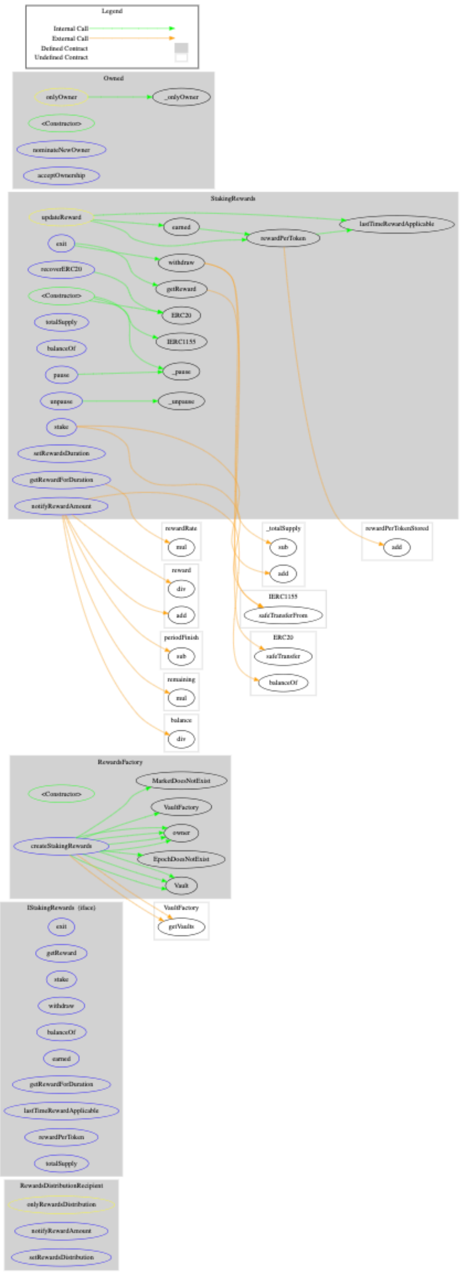
Inheritance Graph

v1.0



CallGraph

v1.0



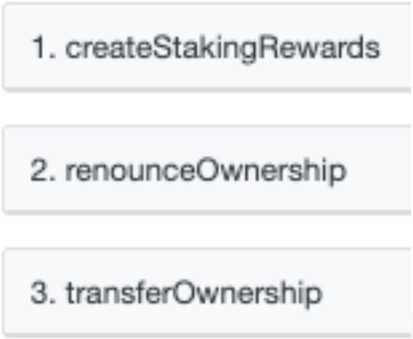
Scope of Work/Verify Claims

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. Overall checkup (Smart Contract Security)

Write functions of contract v1.0



1. createStakingRewards

2. renounceOwnership

3. transferOwnership

Overall checkup (Smart Contract Security)

Tested	Verified
✓	✓

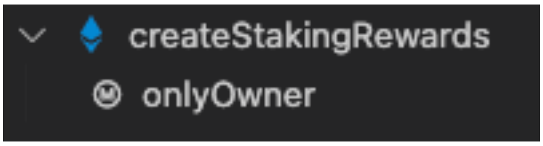
Legend

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

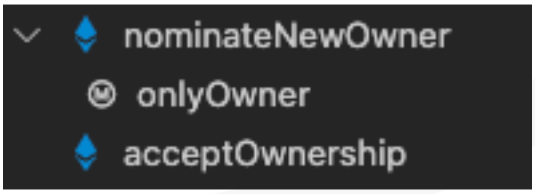
Modifiers and public functions

v1.0

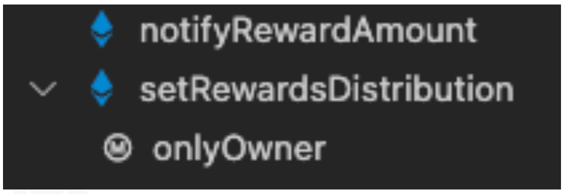
RewardsFactory



```
createStakingRewards
  onlyOwner
```

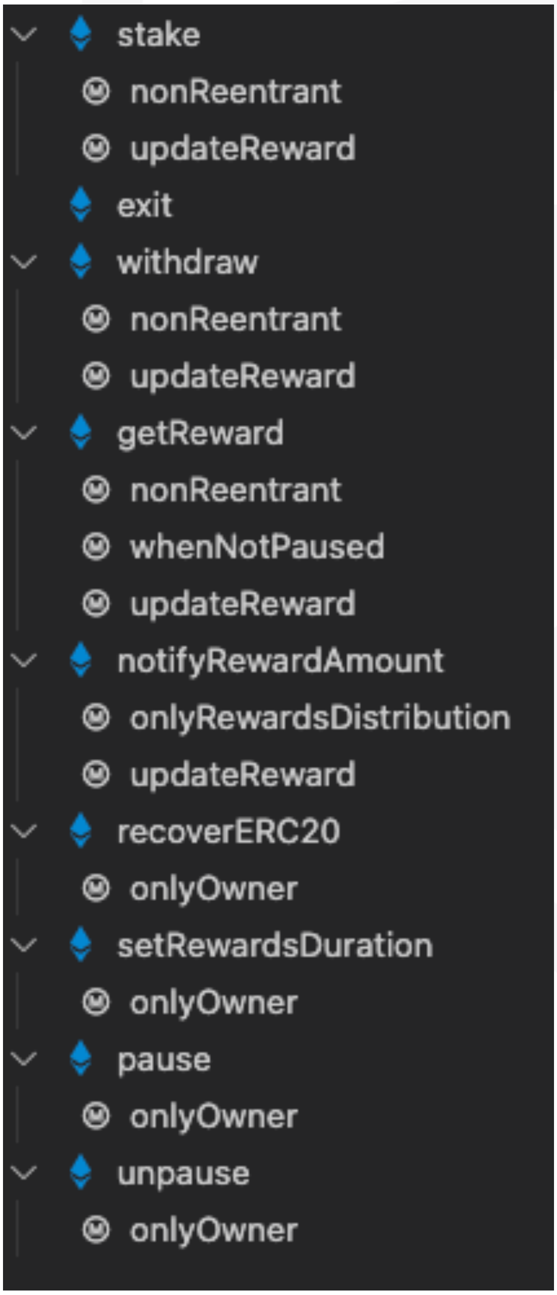


```
nominateNewOwner
  onlyOwner
acceptOwnership
```



```
notifyRewardAmount
setRewardsDistribution
  onlyOwner
```

PausableStakingRewards



```
stake
  nonReentrant
  updateReward
exit
withdraw
  nonReentrant
  updateReward
getReward
  nonReentrant
  whenNotPaused
  updateReward
notifyRewardAmount
  onlyRewardsDistribution
  updateReward
recoverERC20
  onlyOwner
setRewardsDuration
  onlyOwner
pause
  onlyOwner
unpause
  onlyOwner
```

Comments

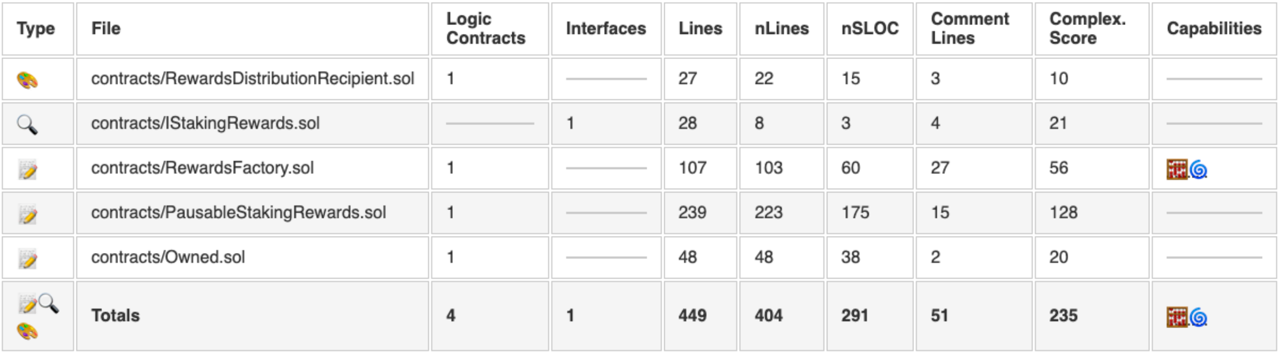
- *Deployer can set following state variables without any limitations*
  - PausableStakingRewards
    - rewardsDuration
- *Deployer can enable/disable following state variables*
- *Deployer can set following addresses*
  - Owned
    - owner
    - nominatedOwner
- *Existing Modifiers*
  - Owned
    - onlyOwner
  - PausableStakingRewards
    - updateReward

- RewardsFactory
  - Owner can create new staking reward contracts
- PausableStakingRewards
  - Owner is able to
    - Pause the contract
    - recoverERC20 tokens
    - Lock the contract by setting “rewardsDuration” to 0. Rewards distributoer are not able to call the following functions afterwards because of dividing by 0 issue
      - notifyRewardAmount

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/RewardsDistributionRecipient.sol	1	----	27	22	15	3	10	----
🔍	contracts/IStakingRewards.sol	----	1	28	8	3	4	21	----
📝	contracts/RewardsFactory.sol	1	----	107	103	60	27	56	🧮🌀
📝	contracts/PausableStakingRewards.sol	1	----	239	223	175	15	128	----
📝	contracts/Owned.sol	1	----	48	48	38	2	20	----
📝🔍🎨	**Totals**	**4**	**1**	**449**	**404**	**291**	**51**	**235**	🧮🌀

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	Owned	Ownership is directly transferred while nominating	See description	We recommend you the remove the L27-L29 because it sets direclty the new ownership. The nominatedOwner must call the "acceptOwnership" to accept the ownership. Additionally we are recommending you to add also the revert nominating function to remove the nominated owner and a renounce Ownership function. Also you can use the openzeppelin Ownable2Step contract

Low issues

Issue	File	Type	Line	Description
#1	Owned	Missing Zero Address Validation (missing-zero-check)	23	Check that the address is not zero
#2	RewardsDistributionRecipient	Missing Zero Address Validation (missing-zero-check)	21	Check that the address is not zero

#3	RewardsFactory	Missing Zero Address Validation (missing-zero-check)	44, 45	Check that the address is not zero
#4	PausableStakingRewards	Missing Zero Address Validation (missing-zero-check)	75, 76, 77	Check that the address is not zero
#5	RewardsDistributionRecipient	Missing Events Arithmetic	25	Emit an event for critical parameter changes

Informational issues

Issue	File	Type	Line	Description
#1	RewardsFactory	State variables that could be declared immutable	12, 13	Add the `immutable ` attributes to state variables that never change
#2	PausableStakingRewards	State variables that could be declared immutable	84	Add the `immutable ` attributes to state variables that never change
#3	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.

21. January 2023:

- 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	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