SOLIDProof
Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC Development | Marketing

MADE IN GERMANY

BlueSale Plugin

Audit

Security Assessment
05. May, 2023

For

BlueSale

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	27. April 2023	• Layout project • Automated- /Manual-Security Testing • Summary
1.1	05. May 2023	• Reaudit

Network
Arbitrum

Website
https://www.bluesale.finance/

Telegram
https://t.me/BlueSaleFinanceGlobal

Twitter
https://twitter.com/BluesaleBls

Discord
https://discord.io/BlueSale

Description
Introducing BlueSale, a decentralized launchpad that allows users to effortlessly launch their own tokens and host initial token sales. You don't need to know how to code. Just go to our terminal and design your own token with a few clicks.
BlueSale has many features that make it easier to launch a token, such as automatic listing on any DEX, LP lock options, and the ability to give your tokens a vesting period.

Project Engagement
During the Date of 24 April 2023, **BlueSale Team** engaged Solidproof.io to audit Plugin 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

BlueSale

Contract Link
v1.0
- https://github.com/BlueSaleOfficial/BlueSale-contracts
- Commit: 950ff86

Note - This Audit report consists of security analysis of the BlueSale smart contracts. Functional testing (or unit testing) of the contract's logic was not included in this analysis.

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.

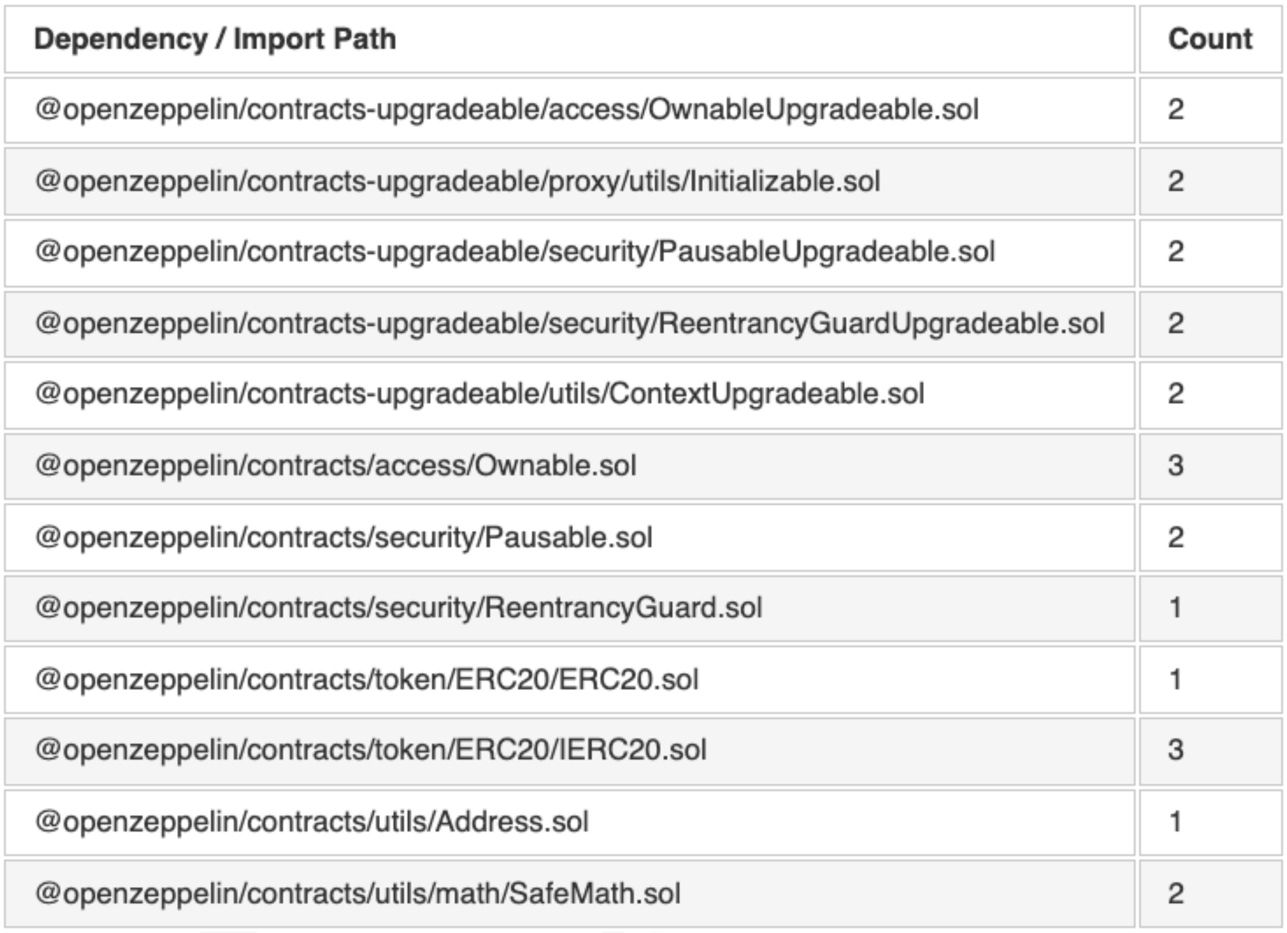
2. Testing and automated analysis that includes the following:
    i) Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
    ii) Symbolic execution, which is analysing a program 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	2
@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol	2
@openzeppelin/contracts-upgradeable/security/PausableUpgradeable.sol	2
@openzeppelin/contracts-upgradeable/security/ReentrancyGuardUpgradeable.sol	2
@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol	2
@openzeppelin/contracts/access/Ownable.sol	3
@openzeppelin/contracts/security/Pausable.sol	2
@openzeppelin/contracts/security/ReentrancyGuard.sol	1
@openzeppelin/contracts/token/ERC20/ERC20.sol	1
@openzeppelin/contracts/token/ERC20/IERC20.sol	3
@openzeppelin/contracts/utils/Address.sol	1
@openzeppelin/contracts/utils/math/SafeMath.sol	2

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.

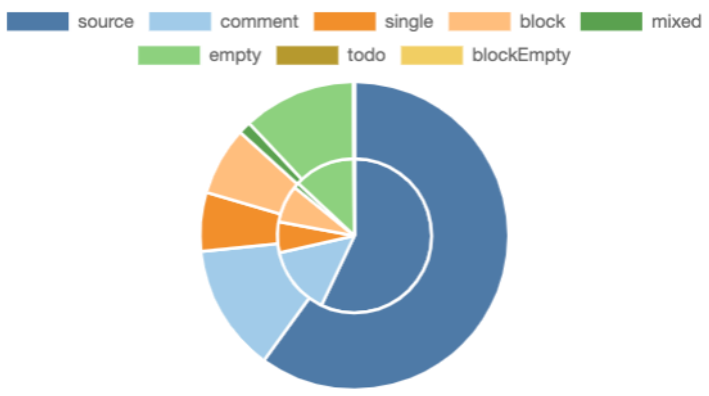
v1.0

File Name	SHA-1 Hash
contracts/RedeemBLS.sol	e6cb68dbfbd0c7fc7d58929b1bcb1717d8555331
contracts/StakeDividendXBLS.sol	f20a234bcbf26779fa232e7c219bd56061656de8
contracts/SwapBLSToXBLS.sol	13f182016eef1e6291e0671dd48520e4ce0ac889

Metrics

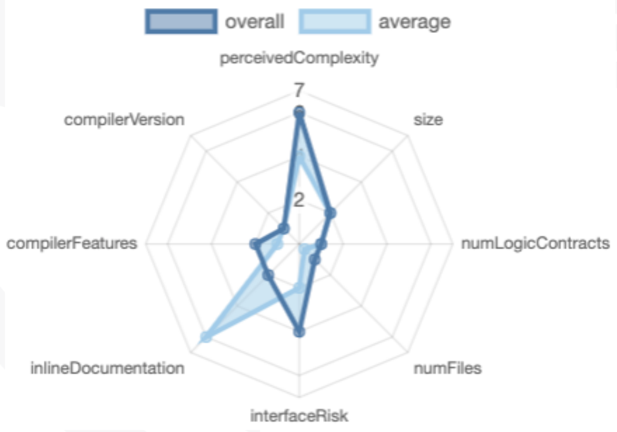
Source Lines

v1.0



Risk Level

v1.0



Capabilities

Components

📝Contracts	📚Libraries	🔍Interfaces	🎨Abstract
3	0	2	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
58	0

External	Internal	Private	Pure	View
49	56	1	0	20

StateVariables

Total	🌐Public
61	60

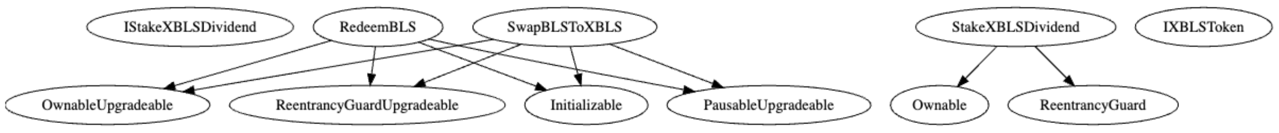
Capabilities

Solidity Versions observed	🧪 Experimental Features	💰 Can Receive Funds	🖥 Uses Assembly	💣 Has Destroyable Contracts
`^0.8.1`				

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

♻️ TryCatch	Σ Unchecked

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. 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 contracts 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

Overall checkup (Smart Contract Security)

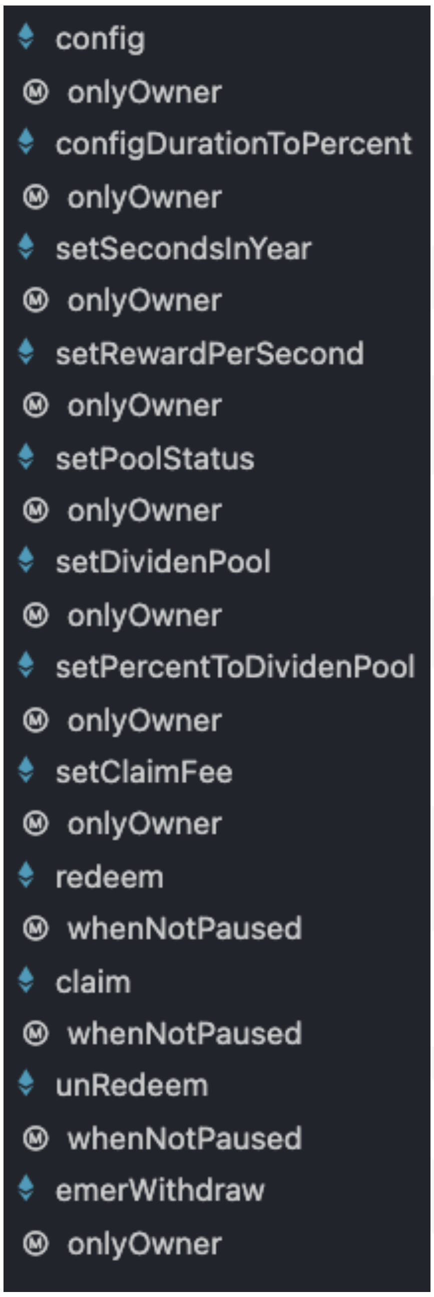
Tested	Verified
✓	✓

Legend

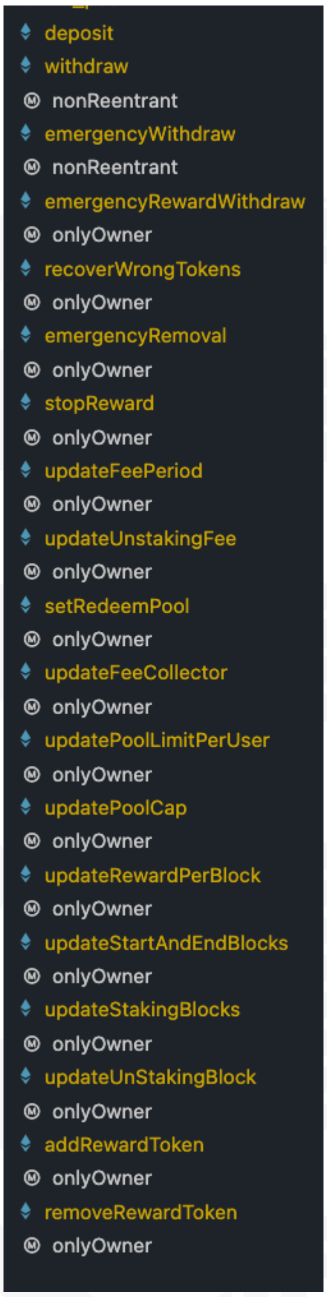
Attribute	Symbol
Verified / Checked	✓
Partly Verified	⚑
Unverified / Not checked	✗
Not available	–

Modifiers and public functions
v1.0

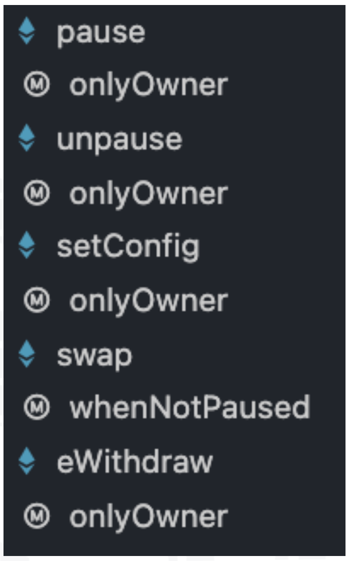
RedeemBLS


config
onlyOwner
configDurationToPercent
onlyOwner
setSecondsInYear
onlyOwner
setRewardPerSecond
onlyOwner
setPoolStatus
onlyOwner
setDividenPool
onlyOwner
setPercentToDividenPool
onlyOwner
setClaimFee
onlyOwner
redeem
whenNotPaused
claim
whenNotPaused
unRedeem
whenNotPaused
emerWithdraw
onlyOwner

StakeDividendXBLS


deposit
withdraw
nonReentrant
emergencyWithdraw
nonReentrant
emergencyRewardWithdraw
onlyOwner
recoverWrongTokens
onlyOwner
emergencyRemoval
onlyOwner
stopReward
onlyOwner
updateFeePeriod
onlyOwner
updateUnstakingFee
onlyOwner
setRedeemPool
onlyOwner
updateFeeCollector
onlyOwner
updatePoolLimitPerUser
onlyOwner
updatePoolCap
onlyOwner
updateRewardPerBlock
onlyOwner
updateStartAndEndBlocks
onlyOwner
updateStakingBlocks
onlyOwner
updateUnStakingBlock
onlyOwner
addRewardToken
onlyOwner
removeRewardToken
onlyOwner

SwapBLSToXBLS


pause
onlyOwner
unpause
onlyOwner
setConfig
onlyOwner
swap
whenNotPaused
eWithdraw
onlyOwner

Ownership Privileges

❖ *RedeemBLS.sol*

- Set reward per second to any arbitrary value including zero
- Set pool start time to any arbitrary value, at any given point in time
- Set/Change dividend pool address (Read Issue#4 for more.)
- Set/Change Claim fee, and percent to dividend pool to any arbitrary value which is not recommended.
- Pause/Unpause the contract and stop its functionality for the users
- Set/Change the following addresses and values
  - Redeem Token(Not Recommended)
  - Reward Token
  - xBLS fee wallet
  - BLS Fee wallet
  - Fee collector(not used anymore in the code)
  - Minimum and Maximum redeem period in a day
  - Percent credit per unit in a day to any arbitrary value

❖ *StakeDividendsXBLS.sol*

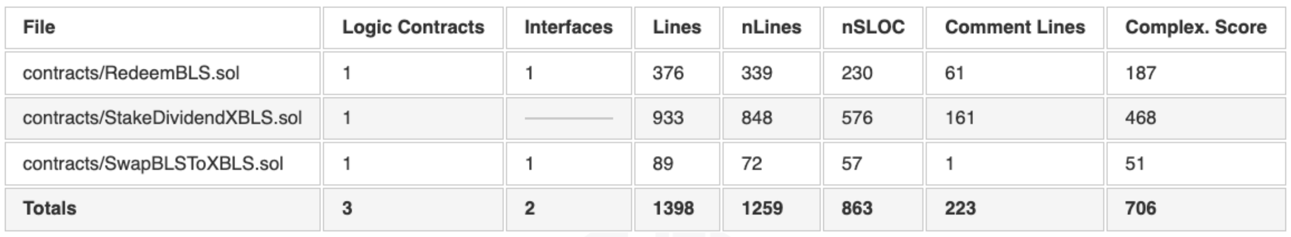
- Recover wrong tokens from the contract, but cannot withdraw staked token
- If the pool is set as a 'removable' at the time of initialisation then the owner will be able to withdraw staked tokens as well.
- Stop Reward
- Update/Set fee period
- Set/Update unstaking fee to any arbitrary value which is not recommended
- Set redeem pool address
- Update pool limit per user to any arbitrary value including zero
- Update reward per block, start and end staking blocks, and start and end blocks of the pool, but not after the pool has started
- Update unstaking block, but not after unstacking has started
- Add/Remove reward token
- Owner can withdraw all the reward tokens from the contract

❖ *SwapBLSToXBLS.sol*

- Set in token, out token, and fund wallet address
- Withdraw any type of tokens from the contract's balance.

Source Units in Scope

v1.0



File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score
contracts/RedeemBLS.sol	1	1	376	339	230	61	187
contracts/StakeDividendXBLS.sol	1	—	933	848	576	161	468
contracts/SwapBLSToXBLS.sol	1	1	89	72	57	1	51
Totals	**3**	**2**	**1398**	**1259**	**863**	**223**	**706**

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	Status
#1	RedeemBLS.sol	Dividend pool percent can be 100% or more	175	The owner can set the dividend pool's percent to 100% or more which is not recommended because it may lead to loss of user funds. Moreover, if it is set to a 100 then the user funds will not be deposited to the contract after redeeming, and all amount will be transferred to the Dividend Pool address which is controllable by the owner. More can be read on issue#4	Acknowledged
#2	RedeemBLS.sol	ClaimFee percent can be 100% or more	179	The owner can set the claim fee percent to 100% or more which is not recommended because it may lead to loss of user funds. Moreover, if it is set to a 100 then the user will not get any amount after claiming, and all amount will be transferred to the Fee wallet address which is controllable by the owner. More can be read on issue#4	Acknowledged

#3	RedeemBLS.sol	Owner can change deposit token	134	User funds can be lost if the owner changes the deposit token address after the users have deposited the tokens. In this scenario the deposited tokens cannot be withdrawn by the users. We recommend to make this address constant or make its change possible only under strict conditions	Acknowledged
#4	StakeDividendXBLS.sol	Unstaking fees can be 100% or more	437	The unstaking fees can be set to 100% or more which is not recommended as this will result in loss of user funds if set to a really high value.	Acknowledged

Low issues

Issue	File	Type	Line	Description	Status
#1	RedeemBLS.sol	Missing Events	155-179	Emit events for critical parameter changes	Acknowledged
#2	RedeemBLS.sol	Missing Zero Address Validation	123-127	Check that the address is not zero	Acknowledged
#3	RedeemBLS.sol	Redundant function	145	The function is redundant and has no role in the logic of the contract. We recommend removing it or adding code into it if necessary	Acknowledged
#4	RedeemBLS.sol	Missing contract validation	171	The contract doesn’t check whether the dividend pool address is a contract or not, and if it is set to an EOA by the owner then while redeeming, the stake amount will go into the EOA of owner’s choosing	Acknowledged

Informational issues

No informational issues

Commented Code exist

There are some instances of code being commented out in the following files that should be removed:

File	Line	Comment
StakeDividendXBLS.sol	257	// _mint(address(msg.sender), _amount);
StakeDividendXBLS.sol	301	// uint256 pending = user.amount.mul(accTokenPerShare).div(PRECISION_FACTOR).sub(user.rewardDebt);
StakeDividendXBLS.sol	393	// require(_tokenAddress != address(rewardToken), "Cannot be reward token");

RedeemBLS.sol	250-252	// if (feeCollector != address(0) && totalharvest > 0) {
		// depositToken.transfer(msg.sender, totalharvest);
		// }

RedeemBLS.sol	108-117	// constructor() {
		// admin = msg.sender;
		// poolStartTime = block.timestamp;
		// yearToSeconds = 31556926;
		// dayToSeconds = 86400;
		// redeemIndex = 1;
		// redeemPeriodMin = 15*dayToSeconds;
		// redeemPeriodMax = 180*dayToSeconds;
		// toDividendPercent = 5000; // 50%
		// }

Recommendation

Remove the commented code, or address them properly.

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.

05. May 2023:

- Unit tests with 100% code coverage was not provided to SolidProof so we cannot ensure complete functional correctness of the code's logic.
- We recommend BlueSale team to conduct unit and fuzz testing thoroughly to rule out possibilities of an unwanted logical and calculation errors.

- There is still an owner (Owner still has not renounced ownership)
- Owner can deploy a new version of the contracts which can change any limit and give owner new privileges due to the upgradeable nature of the contracts.
- The issues found in the v1.0 are acknowledged by the BlueSale team
- 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	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

Solid Proofed

Blockchain Security | Smart Contract Audits | KYC Development | Marketing

MADE IN GERMANY