SOLIDProof
Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC
Development | Marketing

MADE IN GERMANY

0xDragon

Audit

Security Assessment
06. April, 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 security or functionality of the technology we agree to analyze.

Version	Date	Description
1.0	04. April 2023	• Layout project • Automated- /Manual-Security Testing • Summary

Network
Arbitrum

Website
N/A

Twitter
https://twitter.com/0xdragoneth

Description

0xDragon protocol is a force that transcends the ordinary and elevates the potential of LSD assets to new heights. Like a Dragon hoarding its treasure, **this protocol maximizes the return on assets such as stETH, rETH, frxETH, and more, offering a much higher return than typical LSD assets**.
The impact of this protocol is not limited to the Ethereum L1 and L2 realms but ripples across the very fabric of the crypto universe. Its treasury of ETH-related assets grows exponentially, empowering the protocol to hold **a long-term bullish position on ETH.** As the protocol earns real income from Ethereum nodes, its influence only continues to grow.

Project Engagement

During the Date of 04 April 2023, **0xDragon 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://github.com/0xDragoneth/0xDragon
- Commit: 433c1f6

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.

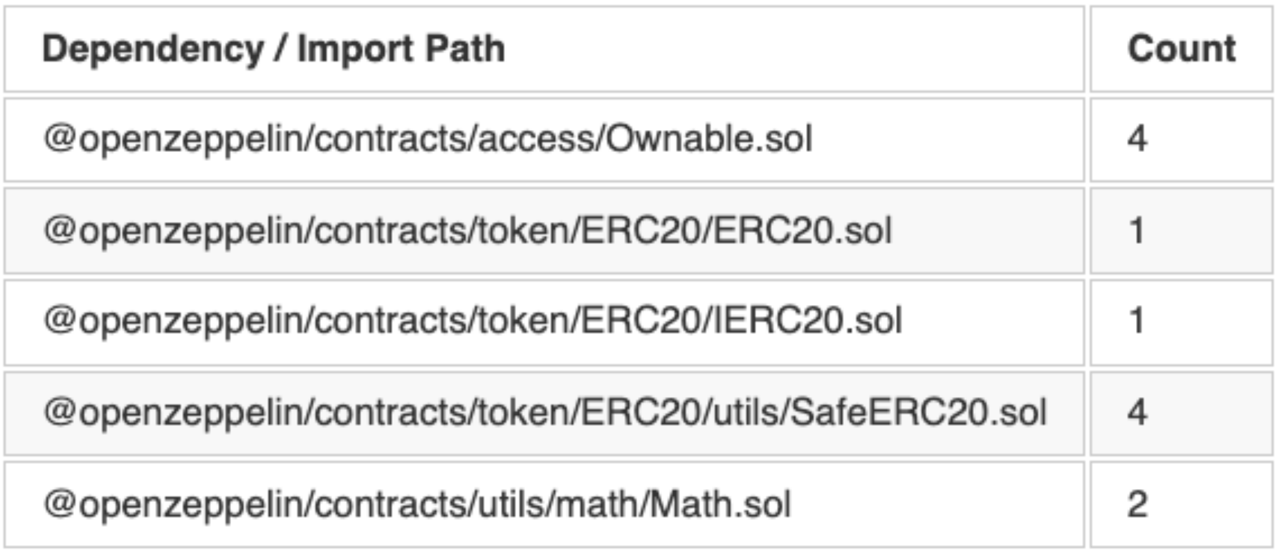
2. Testing and automated analysis that includes the following:
    - i) Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
    - ii) Symbolic execution, which is analysing a program 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	4
@openzeppelin/contracts/token/ERC20/ERC20.sol	1
@openzeppelin/contracts/token/ERC20/IERC20.sol	1
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	4
@openzeppelin/contracts/utils/math/Math.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

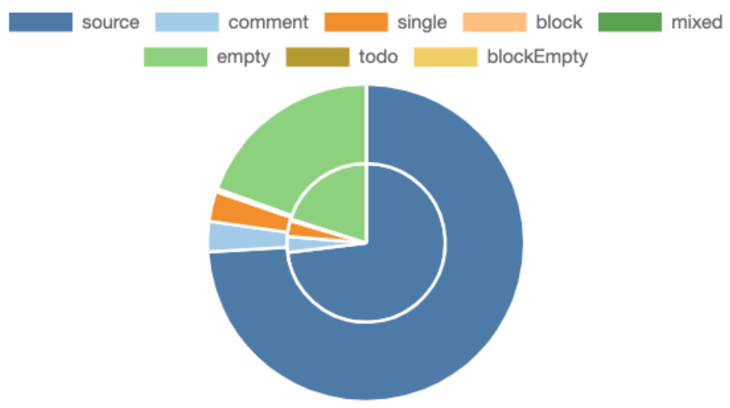
File Name	SHA-1 Hash
contracts/interfaces/IRewardDistributor.sol	2ae309bf5449b6c90d9824f9b5664d5dc70ecbc1
contracts/interfaces/IVirtualBalanceRewardPool.sol	371597866e094c232e4db205c073a0965d754f76
contracts/interfaces/IMintableToken.sol	935ab2de028f1e6ed797fb6db9649324ed659a4a
contracts/interfaces/IRewardTracker.sol	03b43ecbcfa3224937202c0a8b62e7d865560d52
contracts/reward/VirtualBalanceRewardPool.sol	d3d482f907cea46b46c9483b111f29d82e19f7a6
contracts/reward/Vester.sol	54fdf316b338955308dfe80d07c6f3c2801339c8
contracts/reward/RewardDistributor.sol	6d28211421300650a87397acbcdac76572fd058c
contracts/reward/RewardTracker.sol	33b2b38e827b1c08f14e1f83dd2d870f5b0fc7a5
contracts/Lock.sol	55e48ff572269ca3bf84f3809959667dbd137db9
contracts/token/MintableToken.sol	2478aa6d7a4d0552b74bae55ba1282e48c820ce1
contracts/token/BaseToken.sol	83c690cea4b418dc300b374d6303548bb5998bba

contracts/token/esFire.sol	79f761ebb4609b5e728ace31b9fb95099304d3f8
contracts/token/Fire.sol	7fe9a9d526ea595fa69e8961fa3fa008190b4529

Metrics

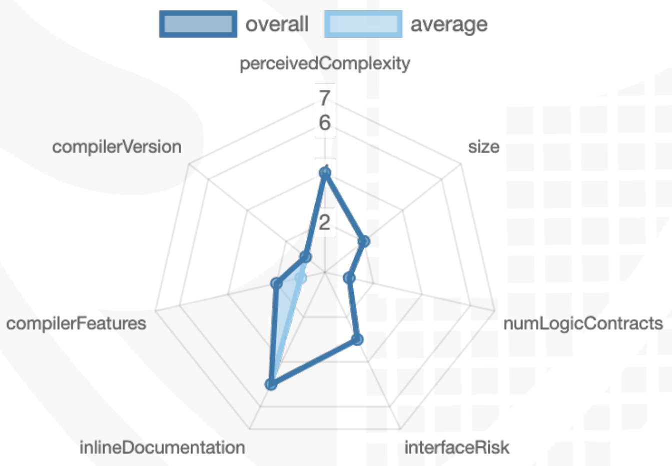
Source Lines

v1.0



Risk Level

v1.0



Capabilities

Components

📝Contracts	📚Libraries	🔍Interfaces	🎨Abstract
7	0	4	2

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
53	1

External	Internal	Private	Pure	View
43	78	7	1	11

StateVariables

Total	🌐Public
54	54

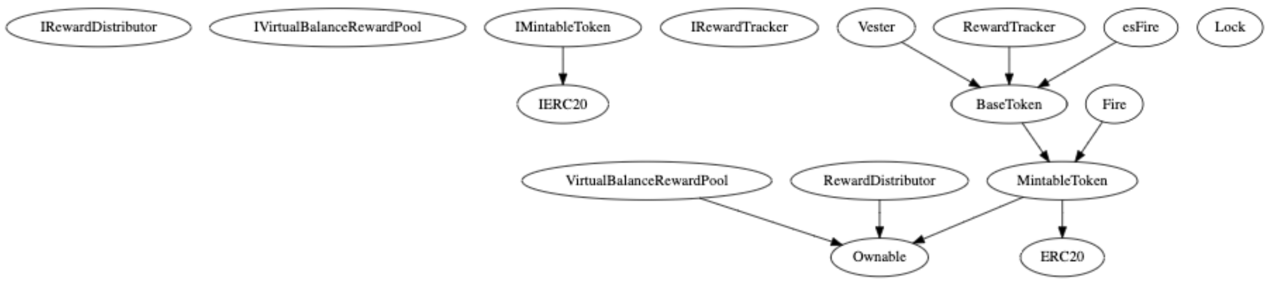
Capabilities

Solidity Versions observed	🧪 Experimental Features	💰 Can Receive Funds	🖥 Uses Assembly	💣 Has Destroyable Contracts
`=0.8.18` `^0.8.9`		`yes`		

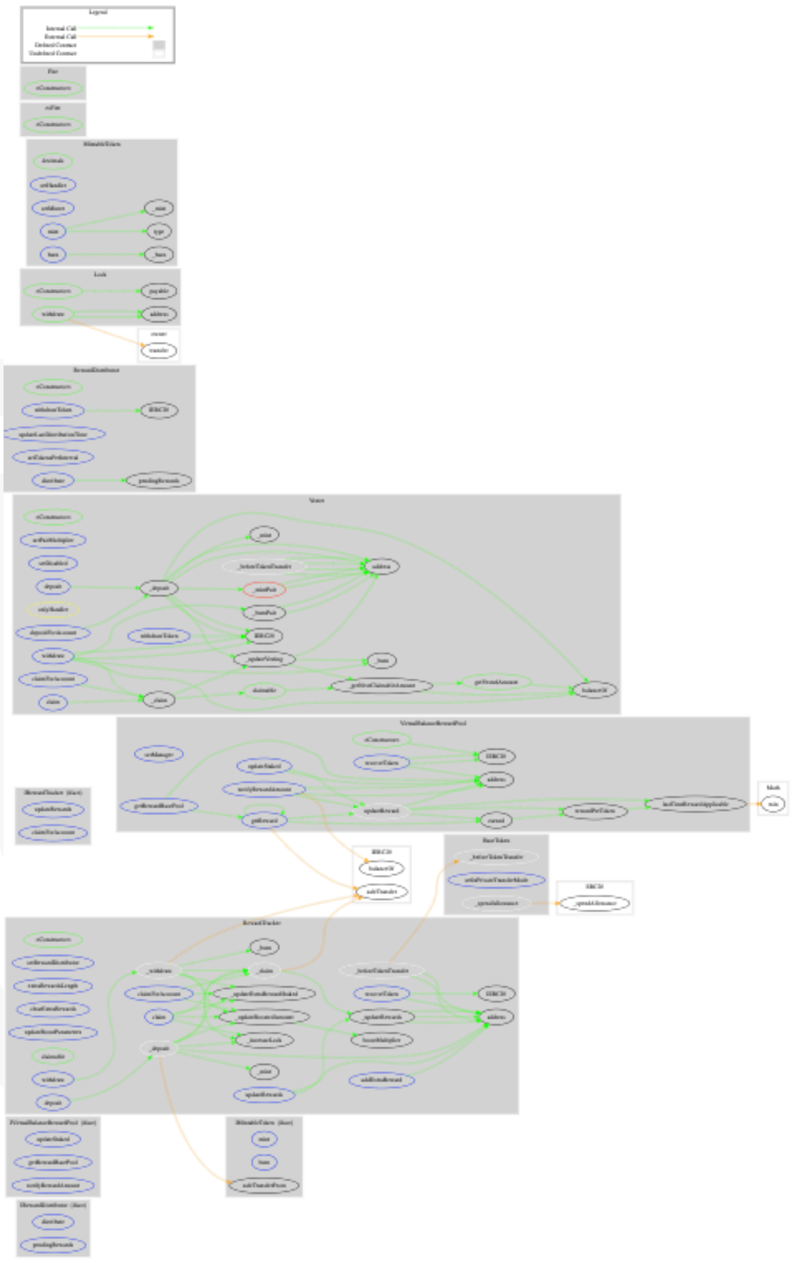
📤 Transfers ETH	⚡ Low-Level Calls	👥 DelegateCall	🧮 Uses Hash Functions	🔖 ECRecover	🌀 New/Create/Create2
`yes`					

♻️ TryCatch	Σ Unchecked
	`yes`

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. 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 can mint	✓	✓	✗
Max / Total Supply	N/A		

Comments:

v1.0

- The minter address set by the owner can mint unlimited tokens

Deployer cannot burn or lock user funds

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

Comments:

v1.0

- Tokens
    - can be burned by the handler addresses
    - can be burned by msg.sender

Deployer cannot pause the contract

Name	Exist	Tested	Status
Deployer can pause	✓	✓	✗

Comments:

v1.0

- Owner can stop deposits in the vesting 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	–	–	–

Overall checkup (Smart Contract Security)

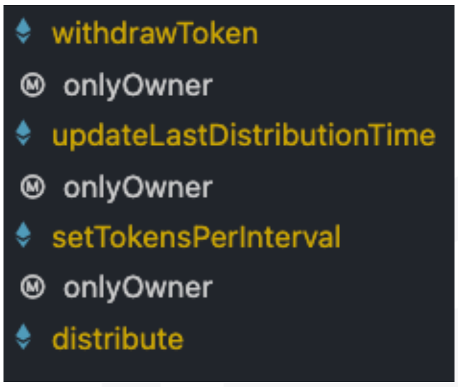
Tested	Verified
✓	✓

Legend

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

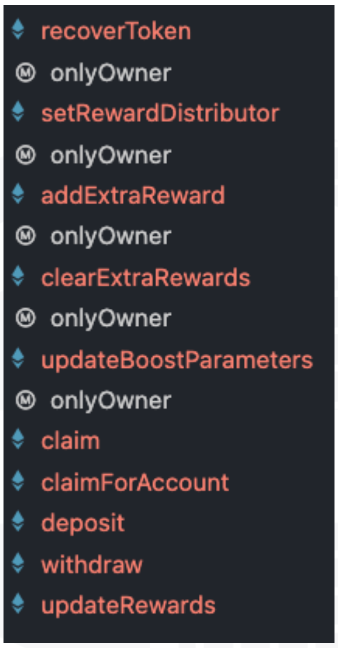
Modifiers and public functions
v1.0

RewardDistributor



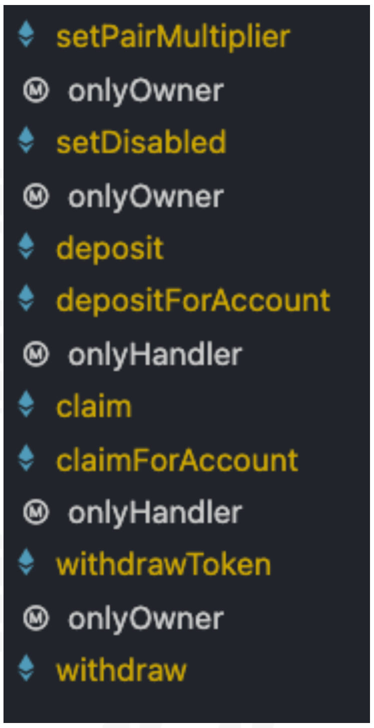
withdrawToken
onlyOwner
updateLastDistributionTime
onlyOwner
setTokensPerInterval
onlyOwner
distribute

RewardTracker



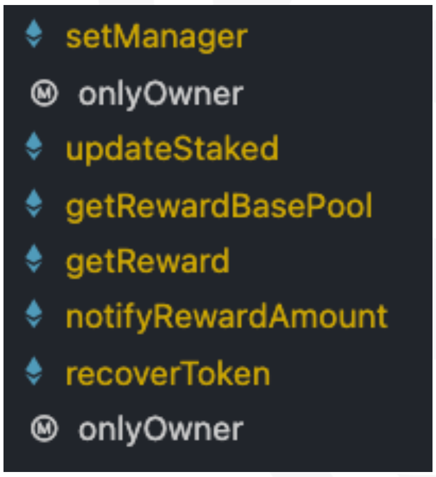
recoverToken
onlyOwner
setRewardDistributor
onlyOwner
addExtraReward
onlyOwner
clearExtraRewards
onlyOwner
updateBoostParameters
onlyOwner
claim
claimForAccount
deposit
withdraw
updateRewards

Vester



setPairMultiplier
onlyOwner
setDisabled
onlyOwner
deposit
depositForAccount
onlyHandler
claim
claimForAccount
onlyHandler
withdrawToken
onlyOwner
withdraw

VirtualBalanceRewardPool



setManager
onlyOwner
updateStaked
getRewardBasePool
getReward
notifyRewardAmount
recoverToken
onlyOwner

Ownership Privileges

- *RewardDistributor.sol*
  - Withdraw tokens from the contract, both the accidentally sent ones and reward tokens.
  - Set tokens per interval

- *RewardTracker.sol*
  - Withdraw any type of tokens from the contract
  - Set/Update reward distributor contract address at anytime
  - Add/Delete extra rewards addresses
  - Update boost parameters to any arbitrary value

- *Vester.sol*

- Set pair multiplier to any arbitrary value including zero and a very high value
- Enable/Disable vesting
- Withdraw vested tokens

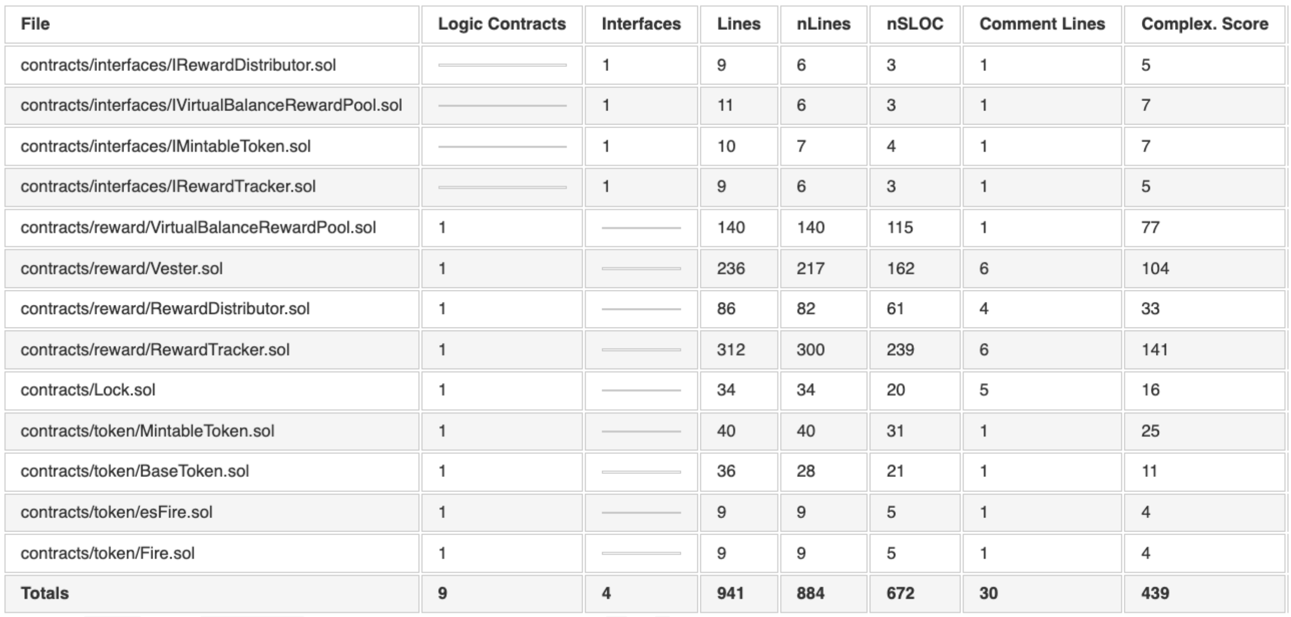
- *VirtualBalanceRewardPool.sol*
  - The base pool address set in the constructor at the time of deployment will be able to change/set the staked balance of a particular account
  - The owner can change the manager address, and the manager address can update the reward rate.
  - Owner can withdraw any type of tokens from the contract including the reward tokens.

- There are several authorities which are authorized to call some functions, that means, if the owner is renounced, another address is still authorized to call functions
  - Be aware of this

Please check if an OnlyOwner or similar restrictive modifier has been forgotten.

Source Units in Scope

v1.0



File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score
contracts/interfaces/IRewardDistributor.sol		1	9	6	3	1	5
contracts/interfaces/IVirtualBalanceRewardPool.sol		1	11	6	3	1	7
contracts/interfaces/IMintableToken.sol		1	10	7	4	1	7
contracts/interfaces/IRewardTracker.sol		1	9	6	3	1	5
contracts/reward/VirtualBalanceRewardPool.sol	1		140	140	115	1	77
contracts/reward/Vester.sol	1		236	217	162	6	104
contracts/reward/RewardDistributor.sol	1		86	82	61	4	33
contracts/reward/RewardTracker.sol	1		312	300	239	6	141
contracts/Lock.sol	1		34	34	20	5	16
contracts/token/MintableToken.sol	1		40	40	31	1	25
contracts/token/BaseToken.sol	1		36	28	21	1	11
contracts/token/esFire.sol	1		9	9	5	1	4
contracts/token/Fire.sol	1		9	9	5	1	4
Totals	**9**	**4**	**941**	**884**	**672**	**30**	**439**

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	RewardDistributor.sol	Owner can drain rewards	33	Owner can withdraw reward tokens from the contract directly.
#2	Vester.sol	Owner can drain toknes	104	There is no check to prevent the owner from withdrawing the vested tokens.
#3	Vester.sol	PairMultiplier must not be zero	67, 151	If the nextPairAMount is higher than the current one only then the pair token will be minted to the caller, and if the PairMultiplier is zero then the pairAMountDiff will cause the function of Deposit to revert.

Low issues

Issue	File	Type	Line	Description
#1	Vester.sol	Missing Zero Address Validation (missing-zero-check)	85, 96	Check that the address is not zero
#2	RewardTracker.sol	Missing Zero Address Validation (missing-zero-check)	80	Check that the address is not zero
#3	MintableToken.sol	Local variables shadowing	20	Rename the local variables that shadow another component
#4	MintableToken.sol	Missing Events Arithmetic	16, 20	Emit an event for critical parameter changes

#5	Vester.sol	Withdraw right after deposit	112	Users can withdraw the vested funds right after depositing them because there is no check or lock period that prevents this from happening.

Informational issues

Issue	File	Type	Line	Description
#1	RewardDistributor.sol	State variables that could be declared immutable (immutable-states)	14, 17	Add the `immutable` attributes to state variables that never change
#2	Vester.sol	Unused return values	225	Ensure that all the return values of the function calls are used and handle both success and failure cases if needed by the business logic
#3	Vester.sol	State variables that could be declared immutable (immutable-states)	12-16	Add the `immutable` attributes to state variables that never change
#4	All	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.

06. April 2023:

- There is still an owner (Owner still has not renounced ownership)
- Owner can deploy a new version of the contract which can change any limit and give owner new privileges
- We recommend the **0xDragon** team to conduct unit tests thoroughly to rule out any logical errors in the contracts.
- It is recommended to add a time lock for vesting
- 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	NOT 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