Security Assessment

StakeWise Staking

Jun 1st, 2021

Summary

This report has been prepared for StakeWise Staking smart contracts, to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Dynamic Analysis, Static Analysis, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases given they are currently missing in the repository;
- Provide more comments per each function for readability, especially contracts are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	StakeWise Staking
Description	The StakeWise smart contracts for tokenized staking and non-custodial validators.
Platform	Ethereum
Language	Solidity
Codebase	https://github.com/stakewise/contracts
Commits	1. 9ec1923bce1d26a7c1dd65f2518080d77f5694da 2. 832333d24eadcb091a82fea9c2f74115a7b19471 3. b8750bd8a6cdd02a371fd15722fdcb2369960808

Audit Summary

Delivery Date	Jun 01, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	Tokenized Staking, Non-Custodial Validators, Staking Rewards, Deposits Pool, Token Vesting

Vulnerability Summary

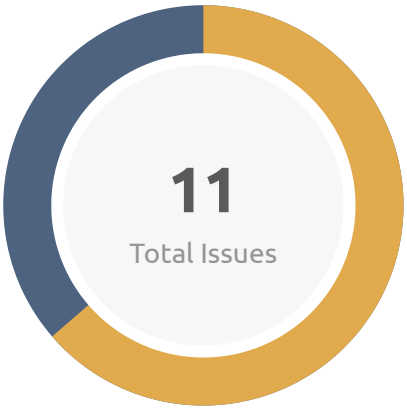
Total Issues	11
Critical	0
Major	0
Minor	7
Informational	4
Discussion	0

Audit Scope

ID	file	SHA256 Checksum
ORA	Oracles.sol	b2a7bb32b2e6d80e89d3989b747ef1ce6e8ab67c59eb7b437101783ba66cdcff
VAL	Validators.sol	f108813305e698b7caedc4d822c29ec6176517f7d8c96138813b653ba44519e3
POO	collectors/Pool.sol	1956804b9f34b0bbfe8c35cd46f0dc0e224e0defcf0da24417a86b08d100d29d
PEE	collectors/PoolEscrow.sol	ab88d3bcf813c2f2c4aaa3eda3e80ab40f4abc965dd1fa5c08a4ecc542b0c756
SOL	collectors/Solos.sol	f5469f2a65e4cfcbebab521c6f716fa351726d6acecdd86f03d08da2b34ad15d
IDC	interfaces/IDepositContract.sol	c15cef89f22a77b9edeb1675304ff8faca573d410f65aa7c20d3d019ebf4d96d
IMD	interfaces/IMerkleDrop.sol	fd1c7d048409e2c337c3c75a00b021d307a8c3b9ee539077cf396d180dce7a88
IOE	interfaces/IOracles.sol	d2487acc23c15864dec69fce7b7c9051c262c9f33310ef1218d59360dfdc68ff
IOP	interfaces/IOwnablePausable.sol	269a87fe98b3581baaae0c7571c8aac5da79ad9658d83155a3a31ebee4df01b6
IPE	interfaces/IPool.sol	16be5394267eb3678a744bbd597954933a2ee7bd303a910fba4f5d507942a867
IPO	interfaces/IPoolEscrow.sol	47e9767eca3626badae5085dea5095eab48686e4a2f707acb46659a53095b017
IRE	interfaces/IRewardEthToken.sol	be7f7d48b3e840ac701059bcc26327a4bcbba85dcf2627d46c5e7a4be9d0e466
ISE	interfaces/ISolos.sol	e16eece416c492f9564dabce9bfbcdc3c4f40efaf7cca1b866ddbaac4fb08eac
IST	interfaces/IStakedEthToken.sol	98982546f4923f4965c6280e92ef5a7a5f45be335c8188b8ee1f64737570c82e
IVE	interfaces/IValidators.sol	b04eeda0e1722975f281106abe99887ea2bf0f52ab87fc90d63918e77219e13c
IVS	interfaces/IVestingEscrow.sol	44984255fead1023d02b8d01ef9af938e11f3d8212202691dd9c6362aabd5c42
IVF	interfaces/IVestingEscrowFactory.sol	2c53320d03c8e2e114fe555e743ce428c802da1dddbf8073e6e15eba5cb2b80e
MDE	merkles/MerkleDrop.sol	321a994c9a71f2ca7584fd01ddf5d46db5a254664f757932e6c4aec722aba3f9
OPE	presets/OwnablePausable.sol	d9af2c6d446e05ff0eca1f2b55e03335a1fde33309ae73e39d563c10695d897d
OPU	presets/OwnablePausableUpgradeable.sol	55565614038f9eece4501ec7c597f5501634dcf6bc009bd118c13d796a0bd98f
ERC	tokens/ERC20PermitUpgradeable.sol	a725cd8015bf51dbe853c2d1d51b67e66170e8f7cb245d4c9f19610f2f2f337f
ERU	tokens/ERC20Upgradeable.sol	d7776c7bafcb61ec38c5e491d5856c6f2734afb6c7f164d8bbd83ef9a76865a2

ID	file	SHA256 Checksum
RET	tokens/RewardEthToken.sol	6d7a669fd90bd6453f074b4779fb3cac21fea45dff45ec8cc57b89b1f261a71f
SWT	tokens/StakeWiseToken.sol	420ba1cbe9bad2b7ba6c4f21c314ad03c60993a3b13e99a5cedccde9db5adf2c
SET	tokens/StakedEthToken.sol	e651614809c75c539188262def752af80c61f40926ec4bac5eb54f351277c114
VEE	vestings/VestingEscrow.sol	ea1a6c2bfa4dc98123805cb7f34fd3c46ada426e81211ff41d874dcc7109f210
VEF	vestings/VestingEscrowFactory.sol	93ecbc203618f05f9fc02888100b581c8a376904f3b7ce8bb350d5d65b17d5d1

Findings



Critical	**0** (0.00%)
Major	**0** (0.00%)
Minor	**7** (63.64%)
Informational	**4** (36.36%)
Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
MDE-01	Potential Over-centralization of Functionality	**Centralization / Privilege**	**Minor**	**Resolved**
ORA-01	Redundant `abi.encodePacked` Utilization	Gas Optimization	Informational	Resolved
ORA-02	Unbounded Sync Period	Logical Issue	Minor	Resolved
PEE-01	Inexistent Input Sanitization	Logical Issue	Minor	Resolved
PEE-02	Pull-Over-Push Pattern	Logical Issue	Minor	Resolved
RET-01	Potential `maintainer`-less Contract	Logical Issue	Minor	Resolved
SOL-01	Redundant `abi.encodePacked` Utilization	Gas Optimization	Informational	Acknowledged
SOL-02	Zero Validator Fees	Logical Issue	Minor	Acknowledged
VEE-01	Inexistent Input Sanitization	Logical Issue	Minor	Resolved
VEE-02	Ambiguous Conditional	Gas Optimization	Informational	Resolved
VEF-01	Redundant `array` Look Up	Gas Optimization	Informational	Resolved

MDE-01 | Potential Over-centralization of Functionality

Category	Severity	Location	Status
Centralization / Privilege	Minor	merkles/MerkleDrop.sol: 84~91	Resolved

Description

The linked function is meant to be used in an edge-case situation whereby the admin is allowed to withdraw the tokens left unclaimed in the airdrop after a specific deadline.

Recommendation

We advise to set a generous deadline to ensure that the normal course of operation of the contract has progressed.

Alleviation

The development team opted to consider our references and commented that the `stop()` function will only be invoked after the token claim period is over and the admin will be the DAO. In addition, the token claim period will be extended as well.

ORA-01 | Redundant `abi.encodePacked` Utilization

Category	Severity	Location	Status
Gas Optimization	Informational	Oracles.sol: 74, 75, 124, 125	Resolved

Description

All variables included in the `abi.encodePacked` invocation cannot be packed under a single 256-bit slot and as such, the invocation is equivalent to `abi.encode` which is more gas efficient. Additionally, when calculating hashes as identifiers it is wise to utilize `abi.encode` instead of `abi.encodePacked` as unaccounted-for tight packs can lead to the same ID being generated with different input variables.

Recommendation

We advise the team to favor utilizing `abi.encode` over `abi.encodePacked`.

Alleviation

The development team opted to consider our references and replaced the linked `abi.encodePacked` instances with `abi.encode` ones.

ORA-02 | Unbounded Sync Period

Category	Severity	Location	Status
Logical Issue	Minor	Oracles.sol: 109~112	Resolved

Description

The `setSyncPeriod()` function fails to check the values of the `_syncPeriod` argument, allowing for either too short or too long period of time.

Recommendation

We advise to add a `require` statement, checking the `_syncPeriod` values against an upper and a lower bound.

Alleviation

The development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase, commenting that admin action will be called after the voting by the DAO.

PEE-01 | Inexistent Input Sanitization

Category	Severity	Location	Status
Logical Issue	Minor	collectors/PoolEscrow.sol: 51~54	Resolved

Description

The `withdraw()` function fails to check the values of the `payee` argument.

Recommendation

We advise to add a `require` statement, checking the `payee` values against the zero address.

Alleviation

The development team opted to consider our references and added the proposed `require` statement, ensuring that the withdrawn Ether from the contract will not be transferred to the zero address.

PEE-02 | Pull-Over-Push Pattern

Category	Severity	Location	Status
Logical Issue	Minor	collectors/PoolEscrow.sol: 42~46	Resolved

Description

The change of admin overrides the previously set admin with the new one without guaranteeing the new admin is able to actuate transactions on-chain.

Recommendation

We advise the pull-over-push pattern to be applied here whereby a new owner is first proposed and consequently needs to accept the owner status ensuring that the account can actuate transactions on-chain.

Alleviation

The development team opted to consider our references and applied the Pull-Over-Push pattern, as proposed.

RET-01 | Potential `maintainer`-less Contract

Category	Severity	Location	Status
Logical Issue	Minor	tokens/RewardEthToken.sol: 75~78	Resolved

Description

The `setMaintainer()` function allows for setting the `maintainer` state variable equal to the zero address, which can lead to token burning in case the `maintainerFee` is not equal to zero.

Recommendation

We advise to add a `require` statement, checking the `_newMaintainer` value against the zero address.

Alleviation

The development team opted to consider our references and added a `require` statement, ensuring inequality of `_newMaintainer` and the zero address.

SOL-01 | Redundant `abi.encodePacked` Utilization

Category	Severity	Location	Status
Gas Optimization	Informational	collectors/Solos.sol: 78, 99, 150	Acknowledged

Description

All variables included in the `abi.encodePacked` invocation cannot be packed under a single 256-bit slot and as such, the invocation is equivalent to `abi.encode` which is more gas efficient. Additionally, when calculating hashes as identifiers it is wise to utilize `abi.encode` instead of `abi.encodePacked` as unaccounted-for tight packs can lead to the same ID being generated with different input variables.

Recommendation

We advise the team to favor utilizing `abi.encode` over `abi.encodePacked`.

Alleviation

The development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase, as the contract is already deployed and is not ugradable.

SOL-02 | Zero Validator Fees

Category	Severity	Location	Status
Logical Issue	Minor	collectors/Solos.sol: 126~129	Acknowledged

Description

The `setValidatorPrice()` function allows for zero validator fees.

Recommendation

We advise to add a `require` statement, introducing an upper and a lower bound to the `validatorPrice` state variable.

Alleviation

The development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase, as the contract is already deployed and is not ugradable.

VEE-01 | Inexistent Input Sanitization

Category	Severity	Location	Status
Logical Issue	Minor	vestings/VestingEscrow.sol: 56~65, 90~101, 106~117	Resolved

Description

The `initialize()`, `stop()` and `claim()` functions fail to check the values of the `address`-type arguments. In the case of `stop()` and `claim()` functions, burning an escrow may be intended functionality.

Recommendation

We advise to add a `require` statement, checking the input values against the zero address.

Alleviation

The development team opted to consider our references and added the proposed `require` statements in `stop()` and `claim()` functions, while also ensuring that the recipient will not be the zero address by adding a `require` statement in the `VestingEscrowFactory` contract, where the vesting escrow will be initialized.

VEE-02 | Ambiguous Conditional

Category	Severity	Location	Status
Gas Optimization	Informational	vestings/VestingEscrow.sol: 75	Resolved

Description

The second part of the linked conditional ambiguously checks the end time against the start time.

Recommendation

We advise to remove the latter part of the linked conditional.

Alleviation

The development team opted to consider our references and optimized the linked conditional.

VEF-01 | Redundant `array` Look Up

Category	Severity	Location	Status
Gas Optimization	Informational	vestings/VestingEscrowFactory.sol: 43	Resolved

Description

The linked `for` loop conditional redundantly uses the `length` member of the specified `array`.

Recommendation

We advise to assign the `array` size to a local variable instead.

Alleviation

The development team opted to consider our references and introduced a local variable to check the `array` size from.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a struct assignment operation affecting an in-memory struct rather than an in-storage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as constant contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK's prior written consent.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing 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. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.