Quantstamp DRAFT

Stakewith.Us

Executive Summary

This audit report was prepared by Quantstamp, the leader in blockchain security.

Type	Staking
Timeline	2023-09-11 through 2023-09-13
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	None
Source Code	• stakewithus/eth-staking-contracts #d960b34
Auditors	• Julio Aguliar Auditing Engineer • Michael Boyle Auditing Engineer • Mustafa Hasan Senior Auditing Engineer

Documentation quality	Medium
Test quality	Low
Total Findings	6 Fixed: 5 Acknowledged: 1
High severity findings	0
Medium severity findings	3 Fixed: 2 Acknowledged: 1
Low severity findings	1 Fixed: 1
Undetermined severity findings	0
Informational findings	2 Fixed: 2

Summary of Findings

The project enables users to stake their ETH and earn rewards from block production, gas tips, and MEV boosts. This is carried out by allowing users to deposit ETH in multiples of 32 plus a fee for deploying validator nodes for the user. A fee recipient contract also gets deployed on the user's first deposit and acts as their reward store from deployed validators.

The code is generally well written and there is a fair amount if inline comments in the Solidity contract files. The provided test suite is comprehensive but lacked good code coverage for multiple contracts. The team spent sometime on improving that and coverage is now above 80% for most contracts over most metrics.

Our assessment did not yield any high severity issues, however a number of medium, low, and informative severity issues were identified and fixed by the team by implementing the recommendations provided by the Quantstamp team.

As for documentation, there is good technical documentation for the project, which provides a good amount of information on how the protocol operates when coupled with the inline code comments. However, user-facing documentation does not seem to exist and we recommend availing such information to end users.

ID	DESCRIPTION	SEVERITY	STATUS
STA-1	Missing Input Validation	Medium	Fixed
STA-2	Privileged Roles and Ownership	Medium	Acknowledged
STA-3	Re-Entrancy Attack Could Drain the `FeeRecipient` Contract	Medium	Fixed
STA-4	`_refund()` Does Not Check if `_validators` Is Zero	Low	Fixed
STA-5	Unlocked Pragma	Informational	Fixed
STA-6	State Changing Functions Do Not Emit Events	Informational	Fixed

Assessment Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

> **Disclaimer**
> Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration in this report.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

1. Code review that includes the following
    1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
    2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
    3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
    1. 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.
    2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Scope

All contracts in the project repository were in-scope for this audit except the `SafeTransferLib.sol` contract.

Files Included

`/src/*`

Files Excluded

`lib/SafeTransferLib.sol`

Findings

STA-1 Missing Input Validation

Medium | Fixed

> **Update**
> Marked as "Fixed" by the client. Addressed in: `0d3e7e515f4a6efb929344048a99b5ef7c849d43`. The client provided the following explanation:
>
> owned.nominateOwner() no zero check is intentional, equivalent to revoking a nomination
>
> Additionally, no maximum was defined for use in `Staking.setOneTimeFee()`.

File(s) affected: `Owned.sol`, `Staking.sol`

Related Issue(s): SWC-123

Description: It is important to validate inputs, even if they only come from trusted addresses, to avoid human error. Following is a list of functions and inputs can could use validation before usage:

1. `Staking.constructor()` should revert if `depositContract_` is the zero address.
2. `Staking.setOneTimeFee()` should revert if `oneTimeFee_` is the same as `oneTimeFee` and does not require the one-time fee to be less than a set maximum.
3. `Staking.setPerformanceFee()` should revert if `performanceFee_` is the same as `performanceFee` and does not require a maximum of less than one hundred percent.
4. `Staking.setTreasury()` should revert if `treasury_` is the zero address or `treasury_` is the same as `treasury`.
5. `Staking.deposit()` should revert if `user_` is the zero address.
6. `Staking.setRefundDelay()` should revert if `refundDelay_` is the same as `refundDelay`.
7. `Owned.nominateOwner()` should revert if `nominatedOwner_` is the zero address.
8. `Owned.constructor()` should revert if `owner` or `operator_` are the zero address.
9. `Owned.setOperator()` should revert if `operator_` is the zero address.

Recommendation: We recommend adding the relevant checks.

STA-2 Privileged Roles and Ownership

Medium | Acknowledged

> **Update**
> Marked as "Acknowledged" by the client. The client provided the following explanation:
>
> > user-facing documentation will be outside of github repo

File(s) affected: `Staking.sol`, `FeeRecipient.sol`

Description: The contracts in this project contain privileged functions that only permitted users can call. These functions control vital aspects of the protocol and abuse of privilege can lead to unexpected behavior and even loss of funds for users. For instance, the owner could set the performance fee to one hundred percent and claim all of the staking rewards for the treasury address that they set.

The following roles have escalated privileges and can perform the following operations:

1. `owner`
   1. Can set the performance fee from zero to one hundred percent.
   2. Can set the one-time fee without any maximum value.
   3. Can set the treasury address to any address.
   4. Can set the refund delay to any length of time greater than or equal to seven days.
2. `operator`
   1. Can call `stake()` to complete the staking process by sending Ether to the deposit contract.
   2. Can refund pending staking deposits earlier than the user can refund themselves.
   3. Can pause and unpause the ability to make deposits. Deposits must be paused and pending validators completed before changing the one-time fee.
3. `treasury`
   1. The treasury address will receive a percentage of staking rewards from all `FeeRecipient` instances as well as the one-time fee for staking.
4. `user`
   1. Can claim their portion of the rewards from their `FeeRecipient` instance.

Recommendation: Make users aware of this via documentation.

STA-3 Re-Entrancy Attack Could Drain the `FeeRecipient` Contract

Medium | Fixed

> **Update**
> Marked as "Fixed" by the client. Addressed in: `a374662acdb17b26c6eb6dc62a14c7d911563958`.

File(s) affected: `FeeRecipient.sol`

Description: The `_treasureClaim()` function transfers `ETH` to the treasury and then updates the `_userRewards` which does not conform with the Check-Effects-Interactions pattern and allows for the possibility of calling `treasureClaim()` multiple times until the contract is drain. The possibility of this happening is very low since we assume the treasure of the project to be a trusted actor behind a multisig. However, in case that multisig gets compromised, this issue would be very severe.

Recommendation: We recommend to follow the CEI pattern by updating `_userRewards` before transferring `ETH` to the treasury.

STA-4 `_refund()` Does Not Check if `_validators` Is Zero

Low | Fixed

Update
Marked as "Fixed" by the client. Addressed in: `ca2080a53423be7ec17bc29be13ab61cc9243a53`.

File(s) affected: `Staking.sol`,

Description: The `_refund()` function proceeds with attempting to transfer the value `validators_ * (_DEPOSIT_AMOUNT + oneTimeFee)` to the `user_` and emit an event even if `_validators` is zero. This leads to unnecessary gas consumption as there is nothing to refund.

Recommendation: Revert if `_validators` is zero.

STA-5 Unlocked Pragma

Informational | Fixed

Update
Marked as "Fixed" by the client. Addressed in: `bb99765e7a5367c2919f93813ceb697a4731beeb`.

File(s) affected: `Staking.sol`, `StakingConstants.sol`, `FeeRecipient.sol`, `lib/*`, `interfaces/*`

Related Issue(s): SWC-103

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.8.*`. The caret (`^`) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence the term "unlocked".

Recommendation: For consistency and to prevent unexpected behavior in the future, we recommend to remove the caret to lock the file onto a specific Solidity version.

STA-6 State Changing Functions Do Not Emit Events

Informational | Fixed

Update
Marked as "Fixed" by the client. Addressed in: `01a2446e160a5b7565607ebc651a79586dee17a2`.

File(s) affected: `FeeRecepient.sol`

Description: Functions `claimRewards()` and `treasuryClaim()` change the contract state and do not emit events. This may be undesirable in case off-chain logic depends on emitted events.

Recommendation: Consider emitting events with the old and new states of the contract.

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
- **Informational** – The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
- **Undetermined** – The impact of the issue is uncertain.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **Acknowledged** – The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

Code Documentation

1. Typo in NatSpec comment for `receive()` : change "if" to "is".

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Contracts

- `ad9...3a6 ./src/Staking.sol`
- `f00...ab8 ./src/FeeRecipient.sol`
- `3d8...ed4 ./src/StakingConstants.sol`
- `6e8...791 ./src/interfaces/IDepositContract.sol`
- `201...4e9 ./src/interfaces/IStaking.sol`
- `c10...5ab ./src/lib/Pausable.sol`
- `7ff...6cd ./src/lib/Owned.sol`
- `47b...7e3 ./src/lib/SafeTransferLib.sol`

Tests

- `6ba...fdd ./test/unit/Staking.t.sol`
- `38d...5ba ./test/unit/FeeRecipient.t.sol`
- `b36...e17 ./test/integration/Staking.integration.t.sol`

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- Slither v0.8.3

Steps taken to run the tools:

1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Automated Analysis

Slither

Slither returned several items, but most were false positives. All the relevant ones were included in this report.

Test Suite Results

```
Running 2 tests for test/unit/Pausable.t.sol:PausableTest
[PASS] test_whenNotPaused() (gas: 35661)
[PASS] test_whenPaused() (gas: 25408)
Test result: ok. 2 passed; 0 failed; 0 skipped; finished in 321.03µs

Running 3 tests for test/unit/Owned.t.sol:OwnedTest
[PASS] test_nominateOwner() (gas: 38404)
[PASS] test_onlyOperator() (gas: 20096)
[PASS] test_setOperator() (gas: 33821)
Test result: ok. 3 passed; 0 failed; 0 skipped; finished in 609.24µs
```

```
Running 12 tests for test/unit/Staking.t.sol:StakingTest
[PASS] test_deposit(uint8) (runs: 256, μ: 519871, ~: 519871)
[PASS] test_deposit_reverts_if_zero_address() (gas: 22493)
[PASS] test_deposit_reverts_on_invalid_amount(uint256) (runs: 256, μ: 30131, ~: 30131)
[PASS] test_deposit_reverts_when_paused() (gas: 27221)
[PASS] test_receive(uint8) (runs: 256, μ: 471470, ~: 471470)
[PASS] test_refund(uint8,uint256) (runs: 256, μ: 449842, ~: 450105)
[PASS] test_refund_reverts_before_refund_delay() (gas: 469941)
[PASS] test_refund_reverts_if_no_pending_validators() (gas: 479692)
[PASS] test_setOneTimeFee_reverts_if_there_are_pending_validators() (gas: 453289)
[PASS] test_setPerformanceFee_cannot_exceed_maximum(uint256) (runs: 256, μ: 13399, ~: 13399)
[PASS] test_setRefundDelay_cannot_exceed_maximum(uint256) (runs: 256, μ: 13464, ~: 13464)
[PASS] test_stake_reverts_if_invalid_length() (gas: 476195)
Test result: ok. 12 passed; 0 failed; 0 skipped; finished in 37.23ms

Running 3 tests for test/unit/FeeRecipient.t.sol:FeeRecipientTest
[PASS] test_claimRewards(uint256) (runs: 256, μ: 93026, ~: 99779)
[PASS] test_claimRewards_reverts_if_nothing_to_claim() (gas: 20416)
[PASS] test_treasuryClaim(uint256) (runs: 256, μ: 99140, ~: 105944)
Test result: ok. 3 passed; 0 failed; 0 skipped; finished in 37.24ms

Running 1 test for test/integration/Staking.integration.t.sol:StakingIntegrationTest
[PASS] test_stake() (gas: 83630)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.24s

Ran 5 test suites: 21 tests passed, 0 failed, 0 skipped (21 total tests)
```

Code Coverage

Coverage for `FeeRecipient.sol` is 100% and for `Staking.sol` is above 80% for lines, statements, and functions, and above 70% for branches. Coverage has been improved for `Owned.sol` and `Pausable.sol` and hits 100% on all criterea.

File	% Lines	% Statements	% Branches	% Funcs
script/Staking.s.sol	0.00% (**0/**6)	0.00% (**0/**8)	0.00% (**0/**4)	0.00% (**0/**2)
src/FeeRecipient.sol	100.00% (**15/**15)	100.00% (**25/**25)	100.00% (**6/**6)	100.00% (**5/**5)
src/Staking.sol	85.71% (**48/**56)	83.33% (**70/**84)	71.88% (**23/**32)	81.25% (**13/**16)
src/lib/Owned.sol	100.00% (**8/**8)	100.00% (**10/**10)	100.00% (**2/**2)	100.00% (**3/**3)
src/lib/Pausable.sol	100.00% (**4/**4)	100.00% (**4/**4)	100.00% (**0/**0)	100.00% (**2/**2)
src/lib/SafeTransferLib.sol	100.00% (**3/**3)	100.00% (**3/**3)	50.00% (**1/**2)	100.00% (**1/**1)
test/unit/Owned.t.sol	100.00% (**1/**1)	100.00% (**1/**1)	100.00% (**0/**0)	100.00% (**1/**1)
test/unit/Pausable.t.sol	100.00% (**2/**2)	100.00% (**2/**2)	100.00% (**0/**0)	100.00% (**2/**2)
test/unit/Staking.t.sol	100.00% (**0/**0)	100.00% (**0/**0)	100.00% (**0/**0)	0.00% (**0/**1)
Total	85.26% (**81/**95)	83.94% (**115/**137)	69.57% (**32/**46)	81.82% (**27/**33)

Changelog

- 2023-09-19 - Initial report

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT

Quantstamp