Quantstamp

# StakeWithUs

## Executive Summary

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

| | |
|---|---|
| Type | EigenLayer Staking |
| Timeline | 2024-05-13 through 2024-05-17 |
| Language | Solidity |
| Methods | Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review |
| Specification | Internal Documentation Provided By Client |
| Source Code | • stakewithus/eigenlayer-staking-contracts #edb1283 |
| Auditors | • Danny Aksenov Senior Auditing Engineer<br>• Jeffrey Kam Auditing Engineer<br>• Valerian Callens Senior Auditing Engineer |

| | |
|---|---|
| Documentation quality | Medium |
| Test quality | Medium |
| Total Findings | 10 Fixed: 4 Acknowledged: 6 |
| High severity findings | 1 Fixed: 1 |
| Medium severity findings | 0 |
| Low severity findings | 6 Fixed: 3 Acknowledged: 3 |
| Undetermined severity findings | 0 |
| Informational findings | 3 Acknowledged: 3 |

## Summary of Findings

Overall, the code adheres to best practices and only one main issue has been identified. Additionally, it's important to note that the security of the `EigenUser.sol` and `EigenStaking.sol` contracts partially depends on the broader EigenLayer protocol and its contracts. Continuous monitoring and review of the EigenLayer protocol's security audits are recommended to ensure ongoing security.

Our testing included manual and automated analysis. While 25 out of 26 tests passed, we observed one failing test. Code coverage was substantial, with `EigenStaking.sol` achieving 85.29% coverage and `EigenUser.sol` at 63.38%. We recommend enhancing branch coverage to at least 90% for better robustness before moving to production.

Update: The StakeWithUs team has either fixed or acknowledged all the issues in the report, as well as significantly improved their test coverage.

| ID | DESCRIPTION | SEVERITY | STATUS |
|---|---|---|---|
| STAKE-1 | Calling `claimETH()` Can Make Future Rewards Unretrievable | High | Fixed |
| STAKE-2 | Partial Exposure to Reentrancies | Low | Fixed |
| STAKE-3 | A Max Limit Should Be Enforced for the Value of `oneTimeFee` | Low | Acknowledged |
| STAKE-4 | Missing Input Validation | Low | Fixed |
| STAKE-5 | Privileged Roles and Ownerships | Low | Acknowledged |
| STAKE-6 | Potential Fee Miscalculation in `claimTokens` | Low | Acknowledged |
| STAKE-7 | Checks-Effects-Interactions Pattern Violation | Low | Fixed |

| ID | DESCRIPTION | SEVERITY | STATUS |
|---|---|---|---|
| STAKE-8 | Unpausing Privilege Could Be Restricted to a More Secure Role | Informational | Acknowledged |
| STAKE-9 | Upgradability | Informational | Acknowledged |
| STAKE-10 | Sender Can Be a Smart Contract without Withdrawal Capabilities | Informational | Acknowledged |

# 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

The actual implementation of the Eigen Layer contracts was out of scope for this audit.

**Files Included**

Repo: https://github.com/stakewithus/eigenlayer-staking-contracts(edb128352e89e1ecf0d9f34456fd725b24e2a998) Files: src/*

**Files Excluded**

Repo: https://github.com/stakewithus/eigenlayer-staking-contracts(58688e758e29e117925bf3c678a0d2a673b98832) Files: lib/*

# Findings

## STAKE-1 Calling `claimETH()` Can Make Future Rewards Unretrievable

• High ⓘ Fixed

**Update**
Marked as "Fixed" by the client.
Addressed in: `39ec299f6f58b831fb5607d5013ed3f6cab14eb6`.

**File(s) affected:** `src/EigenUser.sol`

**Description:** Claiming `claimETH()` sends the full contract's balance to the user. However, the variable `_userETH` is not reset to zero. Thus, whenever `_calculateExecutionRewards()` is called in the future, the following line will revert

```
uint256 balance = address(this).balance - _userETH - _treasuryETH;
```

because `address(this).balance` will be less than `_userETH`. This means `calculateContractETH()`, `claimETH()`, and `treasuryClaim()` will all revert once a user calls `claimETH()`, making future rewards unretrievable for both the operator and the user.

**Recommendation:** Consider setting the variable `_userETH` to zero before the full balance transfer in `claimETH()`.

## STAKE-2 Partial Exposure to Reentrancies

• Low ⓘ Fixed

**Update**
Marked as "Fixed" by the client.
Addressed in: `f32495453dcf38f26df4dd991b5de62c16fafea0`.

**File(s) affected:** `src/EigenStaking.sol`

**Description:** In `EigenStaking`, the functions `refund()`, `stake()`, and `_deposit()` have a `nonReentrant` modifier protecting them from being re-entered. However, we recommend also protecting the system from reentrancies when functions restricted to the operator are executed.

**Recommendation:** In `EigenStaking`, consider adding a `nonReentrant` modifier to the function `stake()` and moving the modifier from `refund()` to `_refund()`.

## STAKE-3 A Max Limit Should Be Enforced for the Value of `oneTimeFee`

• Low ⓘ Acknowledged

**Update**
Marked as "Acknowledged" by the client.
The client provided the following explanation:

```
owner will be a timelock owned by a multisig, so users will have time to respond to fee changes in advance
```

**File(s) affected:** `src/EigenStaking.sol`

**Description:** The value of `oneTimeFee` can be updated to any value via the function `_setOneTimeFee()`, but only when there is no pending validator. While this function can only be executed by the `owner` of the contract, an unlikely scenario with severe consequences could happen:

1. There is no pending validator and `oneTimeFee == 1 ETH`.
2. Bob deposits `330 ETH` to the contract for `10` validators (`10 * (32 + 1)`) via `deposit()`.
3. A compromised owner sees this transaction in the mempool and front-runs the transaction with a call to `setOneTimeFee(298 ETH)`.
4. The transaction of Bob passes, since `330 % (32 + 298) == 0`, and only `1` validator is created.
5. The owner updates the treasury to an address `A` owned by him.
6. The owner updates the operator to an address `B` owned by him.
7. The owner executes `stake()` with address `A` to create one validator for Bob. `298 ETH` are sent to address `B`.

This scenario is considered unlikely since it requires compromising the `owner` of the system.

**Recommendation:** We recommend enforcing a max limit check for the value of `oneTimeFee`.

## STAKE-4 Missing Input Validation

• Low ⓘ Fixed

**Update**
Marked as "Fixed" by the client.
Addressed in: `01a563b3dabdd833fc6629c50ef8ef2482e0deb0`.
The client provided the following explanation:

```
EigenUser is a BeaconProxy deployed via EigenStaking, which checks inputs for initialize. (see line 72
in EigenStaking.sol) deploying the contract directly will not work due to _disableInitializers() in
constructor
  we're not imposing a limit on executionFee and restakingFee for flexibility as we don't know the
potential economic reward structure for the Ethereum network and EigenLayer
```

**File(s) affected:** `src/EigenStaking.sol`, `src/EigenUser.sol`

**Description:** It is important to validate inputs, even if they only come from trusted addresses, to avoid human error:

1. In `EigenStaking.sol`:
   - In the constructor, `treasury_` can be `address(0)` because the non-zero check is done in `setTreasury()` and not `_setTreasury()`;
   - The variables `executionFee` and `restakingFee` can technically be set to `10000`, meaning a fee of `100%`.
2. In `EigenUser.sol`:
   - The `initialize` function is missing input validation for the `user_` and `eigenPodManager_` parameters. This can lead to accidental misconfiguration if these parameters are set to invalid or unintended values.

**Recommendation:** We recommend adding the relevant checks.

## STAKE-5 Privileged Roles and Ownerships

• Low ⓘ Acknowledged

**Update**
Marked as "Acknowledged" by the client.
The client provided the following explanation:

```
documentation will be setup outside of the repo
```

**File(s) affected:** `src/EigenStaking.sol`, `src/EigenUser.sol`

**Description:** All privileged permissions should be clearly documented for users. The privileged permissions of the audited contracts are documented below:

In `EigenStaking.sol`:

- Operator:
  - Via `stake()`, it can stake on behalf of a user. Even if this is compromised, it cannot misappropriate the deposited funds because the number of validators a user is allocated for is tracked by the mapping `pendingValidators`.
  - Via `refundUser()`, it can refund the deposited ETH to a specified user. If compromised, the attacker can refund his own deposited funds without being subjected to the refund delay.
  - Via `pause()` and `unpause()`, it can pause or unpause the `_deposit()` function.
- Owner:
  - It can modify several important variables through `setOneTimeFee()`, `setExecutionFee()`, `setRestakingFee()`, `setTreasury()`, `setRefundDelay()`, and `setImplementation()`. If compromised, it may allow the attacker to steal user's deposits and steal the fees that should be collected by the treasury.
  - By inheriting `Owned.sol`, it can set new operator and nominate new owner through `setOperator()` and `nominateOwner()`.

In `EigenUser.sol`:

- Operator
  - Via `treasuryClaim()`, it can claim the rewards that belong to the treasury.

**Recommendation:** This privileged roles should be made clear to the users, especially depending on the level of privilege the contract allows to the owner.

## STAKE-6 Potential Fee Miscalculation in `claimTokens`

• Low ⓘ Acknowledged

**Update**

Marked as "Acknowledged" by the client.
The client provided the following explanation:

```
this is to handle potential airdrops within the EigenLayer ecosystem - we assume that tokens go:
  eigenPod -> eigenUser -> user
```

**File(s) affected:** `src/EigenUser.sol`

**Description:** The `claimTokens` function in the `EigenUser` contract calculates the fee based on the token balance in the contract. However, if tokens are sent to the contract outside of the `eigenPod.recoverTokens()` function, it could lead to fee miscalculations.

**Recommendation:** Consider updating the fee calculation logic to use the user's specific token balance instead of the contract's total balance.

## STAKE-7 Checks-Effects-Interactions Pattern Violation

• Low Fixed

**Update**
Marked as "Fixed" by the client.
Addressed in: `d82cfbc9a28bc4ef0e6d6a951f98e7aee86ea1c9`.

**File(s) affected:** `src/EigenStaking.sol`, `src/EigenUser.sol`

**Related Issue(s):** SWC-107

**Description:** The Checks-Effects-Interactions coding pattern is meant to mitigate any chance of other contracts manipulating the state of the blockchain in unexpected and possibly malicious ways before control is returned to the original contract. As the name implied, only after checking whether appropriate conditions are met and acting internally on those conditions should any external calls to, or interactions with, other contracts be done. We've identified the following instances, where this pattern has been violated:

1. The `_claimDelayedWithdrawals()` function in the `EigenUser` contract performs an external call to `delayedWithdrawalRouter.claimDelayedWithdrawals()` and then updates the state variables `_userETH`, `_treasuryETH`, and `_queuedTreasuryETH` after the external call.
2. The `_treasuryClaim()` function in the `EigenUser` contract performs an external call to `SafeTransferLib.safeTransferETH()` to transfer ETH to the treasury and then updates the state variables `_userETH` and `_treasuryETH` after the external call.
3. The `_deposit` function in the `EigenStaking` contract updates the storage variables before interacting with an external contract (`BeaconProxy`).

**Recommendation:** We recommend refactoring the code so that it conforms to the Checks-Effects-Interactions pattern and reduces the risk of reentrancy vulnerabilities.

## STAKE-8 Unpausing Privilege Could Be Restricted to a More Secure Role

• Informational Acknowledged

**Update**
Marked as "Acknowledged" by the client.
The client provided the following explanation:

```
as owner will be a timelock/multisig operator has the ability to pause instantly in case of emergency
```

**File(s) affected:** `src/EigenStaking.sol`

**Description:** The functions `pause()` and `unpause()` are restricted to the operator role. Since the action of unpausing the contract is a more powerful action than pausing it (it enables deposits) and should not expected to be used in case of emergency, we suggest restricting it to the owner role which is expected to be owned by more protected and secured addresses.

**Recommendation:** Consider replacing the modifier of the function `unpause()` with `onlyOwner`.

## STAKE-9 Upgradability

• Informational Acknowledged

**Update**

Marked as "Acknowledged" by the client.
The client provided the following explanation:

`documentation will be setup outside of the repo`

**File(s) affected:** `src/EigenUser.sol`

**Description:** While upgradability is not a vulnerability in itself, token holders should be aware that the token contract can be upgraded at any given time. This audit does not guarantee the behavior of future contracts that the token may be upgraded to.

**Recommendation:** The fact that the contract can be upgraded and reasons for future upgrades should be communicated to users beforehand.

### STAKE-10
### Sender Can Be a Smart Contract without Withdrawal Capabilities

• Informational ⓘ Acknowledged

**Update**
Marked as "Acknowledged" by the client.
The client provided the following explanation:

`documentation will be setup outside of the repo`

**File(s) affected:** `src/EigenStaking.sol`

**Description:** The `receive` function in the `EigenStaking` contract allows the sender to be a smart contract without checking if it has withdrawal capabilities. This can lead to funds being stuck in the contract if the sender is a smart contract that cannot handle ETH withdrawals.

**Recommendation:** Inform users interacting with the protocol via a smart contract, that they should implement the appropriate functions for withdrawal of their funds or risk locking their funds.

# 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).

# 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.

**Files**

- `5c5...7a4 ./src/EigenUser.sol`
- `b65...eb6 ./src/EigenStaking.sol`
- `dab...93d ./src/interfaces/IEigenStaking.sol`

- `3d1...396 ./src/interfaces/IEigenUserEvents.sol`
- `4f2...5bf ./src/interfaces/IEigenStakingEvents.sol`
- `e59...fd3 ./src/interfaces/IEigenUser.sol`
- `2a0...e23 ./src/libraries/Owned.sol`
- `509...6a3 ./src/eigenlayer/IDelegationManager.sol`
- `53c...796 ./src/eigenlayer/IEigenPodManager.sol`
- `31f...cdb ./src/eigenlayer/IEigenPod.sol`
- `922...0eb ./src/eigenlayer/IDelayedWithdrawalRouter.sol`

**Tests**

- `5dd...453 ./test/EigenUser.t.sol`
- `cea...e1a ./test/EigenStaking.t.sol`
- `065...019 ./test/helpers/proofs/WithdrawalCredentialsProof.sol`
- `d60...c46 ./test/unit/Owned.t.sol`
- `b44...14c ./test/mocks/MockERC20.sol`

# Toolset

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

**Setup**

Tool Setup:

- Slither v0.10.0

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**

All results identified via slither were either covered by the report or identified as false positives.

# Test Suite Results

All tests are passing.

```
Running 3 tests for test/unit/Owned.t.sol:OwnedTest
[PASS] test_nominateOwner() (gas: 39512)
[PASS] test_onlyOperator() (gas: 20968)
[PASS] test_setOperator() (gas: 35156)
Test result: ok. 3 passed; 0 failed; 0 skipped; finished in 4.18ms

Running 10 tests for test/EigenUser.t.sol:EigenUserTest
[PASS] test_calculateContractETH(uint256) (runs: 256, μ: 25591, ~: 25591)
[PASS] test_claimETH_reverts_if_nothing_to_claim() (gas: 56634)
[PASS] test_claimTokens(uint256) (runs: 256, μ: 666515, ~: 671728)
[PASS] test_claimTokens_reverts_if_nothing_to_claim() (gas: 515150)
[PASS] test_delegation_flow() (gas: 122728)
[PASS] test_queueRestakingRewards(uint256) (runs: 256, μ: 376977, ~: 379748)
[PASS] test_queueRestakingRewards_reverts_if_nothing_to_queue() (gas: 44006)
[PASS] test_treasuryClaim(uint256) (runs: 256, μ: 325984, ~: 332328)
[PASS] test_treasuryClaim_reverts_if_not_operator() (gas: 23697)
[PASS] test_verifyWithdrawalCredentials() (gas: 344375)
Test result: ok. 10 passed; 0 failed; 0 skipped; finished in 1.75s

Running 17 tests for test/EigenStaking.t.sol:StakingTest
[PASS] test_deposit(uint8) (runs: 256, μ: 814497, ~: 814497)
[PASS] test_deposit_reverts_if_zero_address() (gas: 22530)
[PASS] test_deposit_reverts_on_invalid_amount(uint256) (runs: 256, μ: 30221, ~: 30221)
```

```
[PASS] test_deposit_reverts_when_paused() (gas: 778091)
[PASS] test_receive(uint8) (runs: 256, μ: 766452, ~: 766452)
[PASS] test_refund(uint8,uint256) (runs: 256, μ: 742695, ~: 743127)
[PASS] test_refund_reverts_before_refund_delay() (gas: 760617)
[PASS] test_refund_reverts_if_no_pending_validators() (gas: 775996)
[PASS] test_setExecutionFee_cannot_exceed_maximum(uint256) (runs: 256, μ: 13436, ~: 13436)
[PASS] test_setImplementation() (gas: 1865855)
[PASS] test_setImplementation_fails_if_not_contract() (gas: 11448)
[PASS] test_setOneTimeFee_reverts_if_same_value() (gas: 14097)
[PASS] test_setOneTimeFee_reverts_if_there_are_pending_validators() (gas: 746031)
[PASS] test_setRefundDelay_cannot_exceed_maximum(uint256) (runs: 256, μ: 13415, ~: 13415)
[PASS] test_setRestakingFee_cannot_exceed_maximum(uint256) (runs: 256, μ: 13392, ~: 13392)
[PASS] test_stake() (gas: 846914)
[PASS] test_stake_reverts_if_invalid_length() (gas: 775153)
Test result: ok. 17 passed; 0 failed; 0 skipped; finished in 7.39s

Ran 3 test suites: 30 tests passed, 0 failed, 0 skipped (30 total tests)
```

# Code Coverage

We would like to see higher branch coverage for `EigenStaking`. We recommend 90% as a good benchmark prior to going into production.

| File | % Lines | % Statements | % Branches | % Funcs |
|---|---|---|---|---|
| **src/**EigenStaking.sol | 100.00% (**69**/69) | 94.29% (**99**/105) | 82.50% (**33**/40) | 100.00% (**22**/22) |
| **src/**EigenUser.sol | 100.00% (**72**/72) | 100.00% (**103**/103) | 90.91% (**20**/22) | 100.00% (**17**/17) |
| **src/libraries/**Owned.sol | 100.00% (**8**/8) | 100.00% (**10**/10) | 100.00% (**2**/2) | 100.00% (**3**/3) |

# Changelog

- 2024-05-20 - Initial report
- 2024-05-30 - Final 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

**Timeliness of content**

access using the internet or other means, and assumes no obligation to update any information following publication or other making available of the report to you by Quantstamp.

### Notice of confidentiality

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Quantstamp. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Quantstamp.

### Links to other websites

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Quantstamp. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites&aspo; owners. You agree that Quantstamp are not responsible for the content or operation of such web sites, and that Quantstamp shall have no liability to you or any other person or entity for the use of third-party web sites. Except as described below, a hyperlink from this web site to another web site does not imply or mean that Quantstamp endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. Quantstamp assumes no responsibility for the use of third-party software on any website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any output generated by such software.

### Disclaimer

The review and this report are provided on an as-is, where-is, and as-available basis. To the fullest extent permitted by law, Quantstamp disclaims all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. You agree that your access and/or use of the report and other results of the review, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE. This report is based on the scope of materials and documentation provided for a limited review at the time provided. You acknowledge that Blockchain technology remains under development and is subject to unknown risks and flaws and, as such, the report may not be complete or inclusive of all vulnerabilities. The review is limited to the materials identified in the report and does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. The report does not indicate the endorsement by Quantstamp of any particular project or team, nor guarantee its security, and may not be represented as such. No third party is entitled to rely on the report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. Quantstamp does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party, or any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, or any related services and products, any hyperlinked websites, or any other websites or mobile applications, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third party. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.

Quantstamp