Executive Summary

Type: Smart contract
Auditors: Sung-Shine Lee, Research Engineer
          Sebastian Banescu, Senior Research Engineer
          Jake Goh Si Yuan, Security Auditor
Timeline: 2020-07-16 through 2020-08-03
EVM: Muir Glacier
Languages: Solidity
Specification: README.md
Documentation Quality: High
Test Quality: High
Source Code

<table>
<thead>
<tr>
<th>Repository</th>
<th>Commit</th>
</tr>
</thead>
<tbody>
<tr>
<td>easy-staking-contracts</td>
<td>d1f41c</td>
</tr>
<tr>
<td>None</td>
<td>ebecd</td>
</tr>
<tr>
<td>None</td>
<td>724a1</td>
</tr>
</tbody>
</table>

Goals

- Do functions have proper access control logic?
- Are there centralized components which users should be aware of?
- Do the contracts adhere to best practices?
- Are the calculations and funds distribution correct?

Total Issues: 9 (8 Resolved)
High Risk Issues: 1 (1 Resolved)
Medium Risk Issues: 1 (1 Resolved)
Low Risk Issues: 4 (% Resolved)
Informational Risk Issues: 3 (2 Resolved)
Undetermined Risk Issues: 0 (0 Resolved)

- High Risk: The issue puts a large number of users’ sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client’s reputation or serious financial implications for client and users.
- Medium Risk: The issue puts a subset of users’ sensitive information at risk, would be detrimental for the client’s reputation if exploited, or is reasonably likely to lead to moderate financial impact.
- Low Risk: 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.
- Unresolved: Acknowledged the existence of the risk, and decided to accept it without engaging in special efforts to control it.
- 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).
- Resolved: Adjusted program implementation, requirements or constraints to eliminate the risk.
- Mitigated: Implemented actions to minimize the impact or likelihood of the risk.
Summary of Findings

In general, the code is well written, well documented, and well tested. We have, nevertheless, identified one high and one medium severity issue. The high severity issue points out the inadequate implementation of reentrancy guard which still allows reentrancy. The medium refers to the unchecked external calls.

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Severity</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>QSP-1</td>
<td>Reentrancy Guard not implemented properly</td>
<td>♦️ High</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-2</td>
<td>Unchecked external calls</td>
<td>♤ Medium</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-3</td>
<td>Withdrawal Unlock Duration can be set very small</td>
<td>¶️ Low</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-4</td>
<td>Inconsistent use of re-entrancy guard</td>
<td>¶️ Low</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-5</td>
<td>Privileged Roles and Ownership</td>
<td>¶️ Low</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-6</td>
<td>Loss of Precision in Arithmetic Calculations</td>
<td>¶️ Low</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-7</td>
<td>Underspecific claimTokens leads to winner-takes-all</td>
<td>C Informational</td>
<td>Fixed</td>
</tr>
<tr>
<td>QSP-8</td>
<td>Trapped Tokens and Temporary Denial of Service due to overflow of lastDepositIds[address]</td>
<td>C Informational</td>
<td>Acknowledged</td>
</tr>
<tr>
<td>QSP-9</td>
<td>Block Timestamp Manipulation</td>
<td>C Informational</td>
<td>Fixed</td>
</tr>
</tbody>
</table>

Quantstamp Audit Breakdown

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

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

The Quantstamp 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 Quantstamp 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 Quantstamp 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 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, clarify, 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.

Toolset

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

Setup
Tool Setup:

- Slither v0.6.12

Steps taken to run the tools:

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

Findings

QSP-1 Reentrancy Guard not implemented properly

Severity: High Risk

Status: Fixed

File(s) affected: EasyStaking.sol

Description: The `_setLocked` function description indicates that it should prevent reentrancy. However, it does not check whether the lock is being held before and thus diverts from typical mutex implementations. This causes different kinds of problems:

- In the `deposit()` where `_setLocked()` is called, there is no check whether `locked` is set to `true`. Therefore, this method does not stop reentrancy.
- In `onTokenTransfer()`, it checks the lock and only deposits when it is unlocked. Note that it doesn't fail the transaction when the lock is `true`. It is possible to create a mismatch between contracts as the contract that calls this function might think the deposit succeeded.

Recommendation: Use the reentrancy guard from OpenZeppelin.
Update: The team informed us that `setLocked()` is not to guard reentrancy and its main functionality is to avoid the `_deposit()` to be called repeatedly during the ERC677 token transfer. The team updated the comment in `ebecd` and we consider the implementation reasonable.

QSP-2 Unchecked external calls

Severity: Medium Risk

Status: Fixed

File(s) affected: EasyStaking.sol

Description: In calls `token.transfer`, `token.mint` and `token.transferFrom`, the return result from these external calls to `token` are not checked. In case of possible flawed implementation or unthrown failure there can be inconsistent state between `token` and `EasyStaking`. For example, if the `token.transfer()` failed in L448, the balances are still updated in the contract at L437 and it would emit an `Withdrawn` event that has actually failed.

Slither findings:

- `EasyStaking.deposit(uint256,uint256)` (EasyStaking.sol#207-213) ignores return value by `token.transferFrom(msg.sender,address(this),_amount)` (EasyStaking.sol#211)
- `EasyStaking.claimTokens(address,address)` (EasyStaking.sol#282-298) ignores return value by `token.transfer(_to,amount)` (EasyStaking.sol#292)
- `EasyStaking._withdraw(address,uint256,bool)` (EasyStaking.sol#432-450) ignores return value by `token.transfer(liquidityProvidersRewardAddress,feeValue)` (EasyStaking.sol#446)
- `EasyStaking._withdraw(address,uint256,bool)` (EasyStaking.sol#432-450) ignores return value by `token.transfer(_sender,amount)` (EasyStaking.sol#448)
- `EasyStaking._mint(address,uint256,uint256)` (EasyStaking.sol#458-469) ignores return value by `token.mint(address(this),total)` (EasyStaking.sol#463)
- `EasyStaking._mint(address,uint256,uint256)` (EasyStaking.sol#458-469) ignores return value by `token.transfer(liquidityProvidersRewardAddress,total.sub(userShare))` (EasyStaking.sol#466)

Recommendation: Always check the return values of external calls and act accordingly.
Update: The issue is fixed in `ebecd` according to the recommendation.

QSP-3 Withdrawal Unlock Duration can be set very small

Severity: Low Risk

Status: Fixed

File(s) affected: EasyStaking.sol

Description: In `setWithdrawalUnlockDuration()`, if the unlock duration is small, e.g. 1 block, while users can still technically withdraw their funds, in practice, it might be very hard for them to do so.

Recommendation: Consider requiring the duration to be sufficient for end-users to be able to easily withdraw their funds.
Update: The issue is fixed in `ebecd` via adding a requirement so that the unlock duration has to be greater than 1 hour.
QSP-4 Inconsistent use of re-entrancy guard

Severity: Low Risk
Status: Fixed
File(s) affected: EasyStaking.sol

Description: The contract occasionally makes external calls to IERC20Mintable token address. It is assumed that this is the STAKE token contract and is generally trusted. However, in function deposit(uint256, uint256)L207 we see that the mutex is used to protect function call token.transferFrom. Thus the trust model here can be viewed as there may possibly be some reentrancy possibilities from the external calls to token. Under this trust model, all external calls to token should consider the possibility of reentrancy.

There are further non-view calls to token through functions such as token.transfer, token.mint. We suggest using the re-entrancy guard with those calls to be consistent and secure.

Recommendation: Use reentrancy guard in a consistent way.

Update: On ebecd, the original description is related to QSP-1 and thus is considered to be fixed. However, that means that contract as its current form is not protected explicitly by re-entrancy guard. We suggest to implement a proper re-entrancy guard on _withdraw() and _deposit().

Update: On 724a1, the issue is resolved by implementing re-entrancy guard on withdraw() and _deposit().

QSP-5 Privileged Roles and Ownership

Severity: Low Risk
Status: Fixed
File(s) affected: EasyStaking.sol

Description: Smart contracts will often have owner variables to designate the person with special privileges to make modifications to the smart contract. However, this centralization of power needs to be made clear to the users, especially when the owner is given higher level of privileges.

The owner of the EasyStaking contract is able to change several important parameters of the contract repeatedly at any moment in time. These parameters influence:

1. the accrued emissions: setTotalSupplyFactor and setSigmoidParameters
2. the amount of fees that are charged for instant withdrawals: setFee
3. the time when withdrawals with no fees can be performed: setWithdrawalLockDuration and setWithdrawalUnlockDuration
4. the address where liquidity provider rewards (fees) are transferred to: setLiquidityProvidersRewardAddress.

Additionally, the owner can claim unsupported tokens accidentally sent to the contract: claimTokens.

The owner could even front-run end-users by calling functions such as setFee() when an end-user makes a forced withdrawal via makeForcedWithdrawal(). Similar front-running scenarios can happen with the other functions mentioned above as well.

Recommendation: This should be made clear to the end-users via the documentation. Currently, the functions that the owner can call are listed in the README.md, however, the consequences of these functions may not be clear to end-users.

Update: The issue is partially mitigated in ebecd by adding a 7-day period between the request to set a new value and the final setting of this value. Still, we recommend to communicate what the owner can do in the README to completely mitigate the issue.

Update: On 724a1, the team updated the README and resolved the issue.

QSP-6 Loss of Precision in Arithmetic Calculations

Severity: Low Risk
Status: Fixed
File(s) affected: EasyStaking.sol

Description: Solidity integer division might truncate. As a result, performing a multiply before a division might lead to loss of precision. There are 2 occurrences in getAccruedEmission on L397 and L398.

```solidity
    total = _amount.mul(MAX_EMISSION_RATE).div(1 ether).mul(timePassed).div(YEAR);
    userShare = _amount.mul(userEmissionRate).div(1 ether).mul(timePassed).div(YEAR);
```

Recommendation: Move the division after the multiplication.

Update: Fixed in ebecd according to recommendation.

QSP-7 Underspecific claimTokens leads to winner-takes-all

Severity: Informational
Status: Fixed
File(s) affected: EasyStaking.sol

Description: The owner-only function claimTokens is intended to help retrieve tokens and native token sent to the contract address, to be forwarded to a payable address. Currently, the function calculates the amount of tokens/native token to be forwarded by taking the entire balance received unknowingly by the contract address. If this was intentional as the forwarding address is meant to be an intermediary that allowed for further deliberate distribution, then the issue is
no more. However, if not, despite onlyOwner access control to this function, this may lead to some unintentional and intentional flaws.

Exploit Scenario:
1. Alice has sent 10 XDai to the contract. She tries to claim the XDais back to her, and the owner steps in to help.
2. During the time before the owner is able to send a transaction calling claimTokens(address(0), address(Alice)), Bob also sent some XDai to the contract.
3. The owner, not able to distinguish Bob’s transaction before sending out his own, sends out Alice and Bob’s total XDais to Alice.

Recommendation: Set another parameter uint256 _amount to claimTokens.
Update: Fixed in ebecd according to recommendation.

QSP-8 Trapped Tokens and Temporary Denial of Service due to overflow of lastDepositIds[address]

Severity: Informational
Status: Acknowledged
Description: lastDepositIds[address] is used as a way to track the different unique deposits IDs for a given address. In deposit() and withdraw(), the deposit ID 0 is treated as a special case as a validation for wrong ID. Thus funds would be locked if they are deposited with ID 0. At the same time, given that actions of withdraw and deposit require the validation of depositId <= lastDepositIds[address], it can be considered a temporary denial of service if lastDepositIds[address] is set to 0 through overflow. It is only temporary as it can be circumvented by bringing deposits into the account again and increasing lastDepositIds[address].

Exploit Scenario:
1. Increase lastDepositIds[address] to MAXINT(uint256) through deposit() or external token transfer onTokenTransfer().
2. Perform external token transfer onTokenTransfer(), triggering overflow and bringing lastDepositIds[_sender] to 0.

Recommendation: Before allowing ++lastDepositIds[_sender], or pass responsibility to _deposit to validate the for _id the same way as _withdraw, we recommend to perform validation on onTokenTransfer either with SafeMath which would reject further deposits or simply fix the max ID and prevent the ID from overflowing.
Update: The team informed us that "We exclude the possibility of creating such a large number of deposits". We consider this a reasonable assumption.

QSP-9 Block Timestamp Manipulation

Severity: Informational
Status: Fixed
File(s) affected: EasyStaking.sol
Description: Projects may rely on block timestamps for various purposes. However, it’s important to realize that miners individually set the timestamp of a block, and attackers may be able to manipulate timestamps for their own purposes. If a smart contract relies on a timestamp, it must take this into account. Here, the user emission rate is computed based on the block.timestamp, which could be affected by malicious miners.
Recommendation: Add an explicit warning in the end-user documentation indicating that expiration timestamps can have a 900 second error.
Update: This is fixed in ebecd according to the recommendation.

Adherence to Specification
The implementation adheres to the documentation provided.

Code Documentation
The Ethereum code generally adheres to the inline comments and provided documentation. Code comments were included throughout.

Adherence to Best Practices
1. In EasyStaking.sol, _withdraw(), when the _amount is 0, the function withdraws everything for the user. This is not intuitive and may become a source of error if other projects try to integrate with this project. We recommend using MAX_UINT256 as a special value as it is clearer and would not appear in normal calculations.
2. In EasyStaking.sol the following parameters of the event are not indexed:
3. **EasyStaking.sol, L396**: The `require` statement here should be replaced with an `assert` statement, because it is never expected for that invariant to be false. (Fixed in `ebecd`)

4. **EasyStaking.sol, L429**: Should add the comment about how the special case where the function performs differently when `_amount` is 0. (Fixed in `ebecd`)

---

**Test Results**

**Test Suite Results**

<table>
<thead>
<tr>
<th>Contract: EasyStaking</th>
<th>initialize</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓ should be set up correctly (64ms)</td>
<td></td>
</tr>
<tr>
<td>✓ fails if any of parameters is incorrect (1504ms)</td>
<td></td>
</tr>
</tbody>
</table>

| deposit | ✓ should deposit (232ms) |
|         | ✓ should accrue emission (462ms) |
|         | ✓ should deposit using an old id (1846ms) |
|         | ✓ fails if deposit value is zero (65ms) |
|         | ✓ fails if wrong deposit id (62ms) |

| onTokenTransfer | ✓ should deposit (143ms) |
|                | ✓ should accrue emission (598ms) |
|                | ✓ should deposit using an old id (1442ms) |
|                | ✓ fails if deposit value is zero (104ms) |
|                | ✓ fails if not a token address (74ms) |

| makeForcedWithdrawal | ✓ should withdraw (712ms) |
|                      | ✓ should withdraw with accrued emission (335ms) |
|                      | ✓ should withdraw part and accrue emission (315ms) |
|                      | ✓ should accrue emission for different users from 1 address (1554ms) |
|                      | ✓ fails if trying to withdraw more than deposited (392ms) |
|                      | ✓ fails if wrong deposit id (238ms) |
|                      | ✓ fails if zero balance (215ms) |
|                      | ✓ should withdraw entire deposit by several parts (1695ms) |
|                      | ✓ should withdraw the same amount (726ms) |

| requestWithdrawal | ✓ should request (287ms) |
|                  | ✓ fails if wrong deposit id (58ms) |

| makeRequestedWithdrawal | ✓ should withdraw (355ms) |
|                         | ✓ should fail if not requested (140ms) |
|                         | ✓ should fail if too early (226ms) |
|                         | ✓ should fail if too late (199ms) |

| totalStaked | ✓ should be calculated correctly (2235ms) |

| setFee | ✓ should set (97ms) |
|        | ✓ fails if not an owner (53ms) |
|        | ✓ fails if greater than 1 ether (89ms) |

| setWithdrawalLockDuration | ✓ should set (208ms) |
|                           | ✓ fails if not an owner (154ms) |
|                           | ✓ fails if equal to zero (67ms) |

| setWithdrawalUnlockDuration | ✓ should set (112ms) |
|                            | ✓ fails if not an owner (80ms) |
|                            | ✓ fails if equal to zero (50ms) |

| setTotalSupplyFactor | ✓ should set (107ms) |
|                      | ✓ fails if not an owner (76ms) |
|                      | ✓ fails if greater than 1 ether (55ms) |

| setSigmoidParameters | ✓ should set (104ms) |
|                     | ✓ fails if not an owner (50ms) |
|                     | ✓ fails if wrong values (111ms) |

| setLiquidityProvidersRewardAddress | ✓ should set (235ms) |
|                                    | ✓ fails if not an owner (161ms) |
|                                    | ✓ fails if equal to zero (52ms) |
|                                    | ✓ fails if equal to the address of EasyStaking contract (50ms) |

| claimTokens | ✓ should claim tokens (579ms) |
|            | ✓ should claim STAKE tokens (602ms) |
|            | ✓ should claim ether (239ms) |
Code Coverage

The code is well covered by the tests with all the important branches being covered and extensive assertions.

<table>
<thead>
<tr>
<th>File</th>
<th>% Stmts</th>
<th>% Branch</th>
<th>% Funcs</th>
<th>% Lines</th>
<th>Uncovered Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>contracts/</td>
<td>99.3</td>
<td>91.11</td>
<td>100</td>
<td>99.26</td>
<td></td>
</tr>
<tr>
<td>EasyStaking.sol</td>
<td>99.3</td>
<td>91.11</td>
<td>100</td>
<td>99.25</td>
<td>382</td>
</tr>
<tr>
<td>IERC20Mintable.sol</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Sacrifice.sol</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>contracts/lib/</td>
<td>96.67</td>
<td>87.5</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>ExtendedMath.sol</td>
<td>100</td>
<td>87.5</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Sigmoid.sol</td>
<td>94.12</td>
<td>87.5</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>contracts/mocks/</td>
<td>100</td>
<td>75</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>ERC677Mock.sol</td>
<td>100</td>
<td>75</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>EasyStakingMock.sol</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>ExtendedMathMock.sol</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>ReceiverMock.sol</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>All files</td>
<td>98.96</td>
<td>89.47</td>
<td>100</td>
<td>99.45</td>
<td></td>
</tr>
</tbody>
</table>

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

78d0f022f079af06e8071e67758a15e0a17b189dd2ca28f7efde051766aa ./contracts/EasyStaking.sol
01c51557f4f88b8d026d6e84a8c90fe6d19f84688fd1a67789f1a35a09fe6 ./contracts/Sacrifice.sol
4e4d16afe0835d2fe783492ce5f957dca2a2ed4e924f8e7a5157a2ed0818b632f ./contracts/IERC20Mintable.sol
fabca945e52458344ff9d19a3a988e2919bd45b5e59b8cc82fad9a9e9ba394348 ./contracts/lib/ExtendedMath.sol
b9a837c6f4ed730bfc44265fs920ce472f89c621e876829a3c08f6c5df59 ./contracts/lib/Sigmoid.sol
8441319c3d57be08ac5d0a93cb5b8480152358cc6b64be68b8c708e9ba36651 ./contracts/mocks/ERC677Mock.sol
3aeb4b12470f8131631b79db25293bb8a05d885d95644b1a32a090b65a69c ./contracts/mocks/ExtendedMathMock.sol
f418676dc0a0f8125814d3e18e47b76c61da322031453592ee2860beb2a652 ./contracts/mocks/EasyStakingMock.sol
bffa38580954e66d8043f3be47f2f77bd08f6b9740af86993ee5f248728bbb3c7 ./contracts/mocks/ReceiverMock.sol

Tests

49c60ac3946159d1b96ab7b80fe4e403599ed62d24522fc5abaeeea7fad918a10c ./test/EasyStaking.test.js

Changelog
About Quantstamp

Quantstamp is a Y Combinator-backed company that helps to secure blockchain platforms at scale using computer-aided reasoning tools, with a mission to help boost the adoption of this exponentially growing technology.

With over 1000 Google scholar citations and numerous published papers, Quantstamp’s team has decades of combined experience in formal verification, static analysis, and software verification. Quantstamp has also developed a protocol to help smart contract developers and projects worldwide to perform cost-effective smart contract security scans.

To date, Quantstamp has protected $5B in digital asset risk from hackers and assisted dozens of blockchain projects globally through its white glove security assessment services. As an evangelist of the blockchain ecosystem, Quantstamp assists core infrastructure projects and leading community initiatives such as the Ethereum Community Fund to expedite the adoption of blockchain technology.

Quantstamp’s collaborations with leading academic institutions such as the National University of Singapore and MIT (Massachusetts Institute of Technology) reflect our commitment to research, development, and enabling world-class blockchain security.

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by Quantstamp; however, Quantstamp does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication.

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, Inc. (Quantstamp). Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites’ 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 the website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

Disclaimer

This report is based on the scope of materials and documentation provided for a limited review at the time provided. Results may not be complete nor inclusive of all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Blockchain technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. A report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, we disclaim 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. We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, 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, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. 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. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR REPLIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.