Quantstamp

Venus Protocol V2

Executive Summary

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

Type	Peg Stability Module
Timeline	2023-07-24 through 2023-08-07
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	PSM.pdf
Source Code	• VenusProtocol/venus-protocol #a9c6e07
Auditors	• Jonathan Mevs Auditing Engineer • Faycal Lalidji Senior Auditing Engineer • Valerian Callens Senior Auditing Engineer • Cameron Biniamow Auditing Engineer

Documentation quality	High
Test quality	High
Total Findings	6 Fixed: 1 Acknowledged: 5
High severity findings	0
Medium severity findings	0
Low severity findings	4 Fixed: 1 Acknowledged: 3
Undetermined severity findings	0
Informational findings	2 Acknowledged: 2

Summary of Findings

Quantstamp performed an audit of Venus Protocol's Peg Stability Module (PSM). This module is inspired by MakerDAO's PSM for DAI and aims to peg the VAI token to a value of $1. The team plans to deploy two separate PSMs, one whose peg will be maintained by USDC and another whose peg will be maintained by USDT. Users can swap USDC or USDT to VAI and vice versa through these contracts. Fees from each swap are directed to the Venus Treasury. Overall, the codebase is well-written and the Venus team was attentive to security best practices.

This implementation relies heavily on Venus' Resilient Price Oracle which was out of the scope of this audit. Additionally, centralized aspects are present as this contract is upgradable and privileged addresses can modify key state variables of this contract. The tests for Peg Stability were present and thorough.

Fix Review Update
During the Fix Review the team provided details supporting their acknowledgment of findings. Additionally, they removed the ability of the owner to renounce ownership. All issues have been thoroughly addressed.

ID	DESCRIPTION	SEVERITY	STATUS
VEN-1	Uncapped Fee Range	Low	Acknowledged
VEN-2	Ownership Can Be Renounced	Low	Fixed
VEN-3	Privileged Roles and Ownership	Low	Acknowledged
VEN-4	Checks-Effects-Interactions Pattern Violation	Low	Acknowledged
VEN-5	Upgradability	Informational	Acknowledged
VEN-6	Outdated Solidity Version	Informational	Acknowledged

ID	DESCRIPTION	SEVERITY	STATUS

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. This audit only included the file `PegStability.sol`.

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.

Findings

VEN-1 Uncapped Fee Range

Low | Acknowledged

> **Update**
> Marked as "Acknowledged" by the client. The client provided the following explanation:
>
> > Setting of fees will go through governance, so community will be able to vote on the fees. So we don’t think that we should have a tighter bound in the code.

File(s) affected: `PegStability.sol`

Description: The only bounds included for the `feeIn` or `feeOut` are that they do not meet or exceed 100%. This means the fees can be assigned high values such as 99%, which would result in a poor user experience.

Recommendation: Consider whether these fees should have a tighter bound.

VEN-2 Ownership Can Be Renounced

Low | Fixed

Update

Marked as "Fixed" by the client. Addressed in: `dc4ce6fe67198f10527c2f70d943c21adcd8725f`. The client overrode the `renounceOwnership()` function so it is no longer possible to do so.

File(s) affected: `PegStability.sol`

Description: If the owner renounces his ownership, the contract will be left without an owner. Consequently, it will no longer be possible to execute functions guarded by the `onlyOwner` modifier.

Recommendation: Confirm that this is the intended behavior. If not, override and disable the `renounceOwnership()` function.

VEN-3 Privileged Roles and Ownership

Low | Acknowledged

Update

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

> The owner of the PSM contract will be 0×939bd8d64c0a9583a7dcea9933f7b21697ab6396, that is the Timelock contract used to execute the normal Venus Improvement Proposals (VIP). For normal VIPs, the time config is: 24 hours voting + 48 hours delay before the execution. So, only the community, via a VIP will be able to execute the mentioned protected functions. We'll use the AccessControlManager (ACM) deployed at https://bscscan.com/address/0×4788629abc6cfca10f9f969efdeaa1cf70c23555 In this ACM, only 0×939bd8d64c0a9583a7dcea9933f7b21697ab6396 (Normal) has the DEFAULT_ADMIN_ROLE. And this contract is a Timelock contract used during the Venus Improvement Proposals. The idea is to grant 0×939bd8d64c0a9583a7dcea9933f7b21697ab6396 to execute every mentioned function. Moreover, we'll allow a and b also to execute the following functions:
>
> - pause()
> - resume() Specifically, the current config for the three Timelock contracts are:
> - normal: 24 hours voting + 48 hours delay
> - fast-track: 24 hours voting + 6 hours delay
> - critical: 6 hours voting + 1 hour delay [a] https://bscscan.com/address/0×555ba73dB1b006F3f2C7dB7126d6e4343aDBce02 [b] https://bscscan.com/address/0×213c446ec11e45b15a6E29C1C1b402B8897f606d

File(s) affected: `PegStability.sol`

Description: Smart contracts often have operations that can only be executed by a limited amount of addresses with special privileges. The access control mechanism of the current contract is enforced using the contract `AccessControlledV8` and the function `_checkAccessAllowed()`.

The contract `PegStability.sol` gives the following privileges:

1. Any authorized address can pause (function `pause()`) and unpause (function `resume()`) the contract. Swaps are only possible when the contract is not paused.
2. Any authorized address can update the storage variables:
   - `feeIn` and `feeOut` within the range `0` to `9999`, which means a fee of `0%` to `99.99%`;
   - `vaiMintCap` which represents the maximum amount of VAI that can be minted by this contract;
   - `venusTreasury` which represents the beneficiary address of swap fees;
   - `oracle` which represents the address giving the current price of the stable token in USD.

Note that the access control mechanism is currently managed via governance decisions.

Recommendation: This centralization of power needs to be made clear to the users, especially depending on the level of privilege the contract allows to the authorized address.

VEN-4 Checks-Effects-Interactions Pattern Violation

Low | Acknowledged

Update

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

> The external calls are to known contracts and here is the reason CEI pattern is violated:
>
> 1. Stabletoken.transferFrom(): in order to support fee on transfer we have to call transfer before, so we can calculate the actual transferred amount
> 2. oracle.updateAssetPrice(): before calculating the USD value of the asset we need to update the assetPrice, to make sure we are not reading an outdated price. Moreover this contract is known and is created by us, so no unexpected control of the flow can happen. On top of that we are using nonReentrant modifier for both swapVaiForStable and swapStableForVai. Moreover We will definitely do a due Diligence on the stable tokens, and initially we will start with

only USDT which does not implement any hooks and is non-upgradeable
https://bscscan.com/token/0×55d398326f99059ff775485246999027b3197955#code

File(s) affected: `PegStability.sol`

Related Issue(s): SWC-107

Description: If non-reentrant modifiers are used without adhering to the CEI pattern, it cannot guarantee that the global system state is reentrancy-free. There are schemes that can be exploited on a larger scale. All external calls must be executed after the state has been updated because multiple other contracts may depend on the current PegStability contract state. This violation is currently present in `swapStableForVAI()`.

Recommendation: Two options exist to ensure the safety of `swapStableForVAI()` function:

1. Guarantee that all externally called functions in `swapStableForVAI()` do not contain hooks that might trigger a call to a third contract that uses the `PegStability` contract state.
2. Alternatively, move all external calls to the end of the function execution. This way, the `PegStability` contract state will be updated before any external call occurs, preventing any unexpected interactions with external contracts.

VEN-5 Upgradability

• Informational Acknowledged

Update

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

> Upgrading implementation goes again through governance, so unexpected upgrade of the implementation is not possible, unless the governance vote passes through.

File(s) affected: `PegStability.sol`

Description: While upgradability is not a vulnerability in itself, users should be aware that the `PegStability` implementation contract can be upgraded. This audit does not guarantee the behavior of future implementation contracts that `PegStability` may be upgraded to. Here are two examples of what could contain a malicious upgraded contract:

- A function draining all stable tokens owned by the contract to an arbitrary address.
- A function minting any amount of VAI (because the proxy address is authorized to mint VAI).

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

VEN-6 Outdated Solidity Version

• Informational Acknowledged

Update

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

> The team is aware that versions should be bumped to a higher one. However in the case of PSM, it is dependent on ResilientOracleInterface which version is 0.8.13, and we would need to first update Oracle version and then update PSM. This will be done in the future for all Venus Contracts for which update of Solidity version is possible.

File(s) affected: `PegStability.sol`

Description: As security standards develop, so does the Solidity language. In order to stay up to date with current practices, it's important to use a recent version of Solidity and recent conventions.

Recommendation: Consider using solidity version `0.8.18` instead of `0.8.13` and refer to the list of recommended versions for up-to-date suggestions.

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. There is a mismatch between the documentation mentioning the variables `tin` and `tout` to store the value of the fees for each swap direction, and the variables `feeIn` and `feeOut` used in the contract.
2. Consider rewording the enum `FeeDirection` in `StableDirection` to make it more clear.

Adherence to Best Practices

1. `previewTokenUSDAmount()`, `getPriceInUSD()`, `ensureNonzeroAddress()` can be renamed to being with underscores to follow internal & private function naming conventions.
2. Consider adding an `I` at the beginning of each name of the interfaces.
3. For consistency, consider always using `Vai` or `VAI` in the names of functions and events.
4. For consistency, consider using the same order of fields for the events `StableForVAISwapped` and `VaiForStableSwapped`, meaning "in, out, fee" or "in, fee, out".
5. It is possible to decrease the size of the contract and ultimately reduce the deployment cost (in gas) by reducing the size of the messages of the require statements. For instance, the usage of a `.` at the end of the message could be removed.
6. For gas optimization, it is possible to use the operator != 0 instead of > 0 when dealing with a uint256 variable to reduce the gas cost of asserting the conditional statement.
7. Change the type of `VAI_ADDRESS` to `VAI` from `address` instead of casting `VAI_ADDRESS` to `VAI` each time it is referenced in the contract.

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

- `76e...c89 ./PegStability.sol`

Tests

- `1c2...47a ./VAI/PegStability.ts`

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

We have executed slither and filtered the issues that were reported and incorporated the valid ones in the report. Please note that only issues related to the scope of the audit are reported.

Test Suite Results

Tests were run with `npx hardhat test ./tests/hardhat/VAI/PegStability.ts`.

Update
Some tests were renamed and all tests are still passing.

```
Peg Stability Module
  PSM: 18 decimals
    initialization
      ✔ should revert if contract already deployed
      ✔ should initialize sucessfully
      reverts if init address = 0x0:
        ✔ acm
        ✔ treasury
        ✔ stableToken
      reverts if fee init value is invalid
        ✔ feeIn
        ✔ feeOut
    Admin functions
      pause()
        ✔ should revert if not authorised
        ✔ should pause if authorised
        ✔ should revert if already paused
      resume()
        ✔ should revert if not authorised
        ✔ should resume if authorised
        ✔ should revert if already resumed
      setFeeIn(uint256)
        ✔ should revert if not authorised
        ✔ should revert if fee is invalid
        ✔ set the correct fee
      setFeeOut(uint256)
        ✔ should revert if not authorised
        ✔ should revert if fee is invalid
        ✔ set the correct fee
      setVAIMintCap(uint256)
        ✔ should revert if not authorised
        ✔ should set the correct mint cap
      setVenusTreasury(uint256)
        ✔ should revert if not authorised
        ✔ should revert if zero address
        ✔ should set the treasury address
      setOracle(address)
        ✔ should revert if not authorised
        ✔ should revert if oracle address is zero
        ✔ should set the oracle
    Pause logic
      ✔ should revert when paused and call swapVAIForStable(address,uint256)
      ✔ should revert when paused and call swapStableForVAI(address,uint256)
    Swap functions
      swapVAIForStable(address,uint256)
        ✔ should revert if receiver is zero address
        ✔ should revert if sender has insufficient VAI balance
        ✔ should revert if VAI transfer fails
        ✔ should revert if VAI to be burnt > vaiMinted
        should sucessfully perform the swap
          Fees: 10%
            ✔ stable token = 1$
            ✔ stable token < 1$
            ✔ stable token > 1$
          Fees: 0%
            ✔ stable token = 1$
            ✔ stable token < 1$
```

```
        ✔ stable token > 1$
    swapStableForVAI(address,uint256)
      ✔ should revert if receiver is zero address
      ✔ should revert if VAI mint cap will be reached
      ✔ should revert if amount after transfer is too small
      should sucessfully perform the swap
        Fees: 10%
          ✔ stable token = 1$  (39ms)
          ✔ stable token > 1$  (38ms)
          ✔ stable token < 1$  (39ms)
        Fees: 0%
          ✔ stable token = 1$
          ✔ stable token > 1$
          ✔ stable token < 1$
PSM: 8 decimals
  initialization
    ✔ should revert if contract already deployed
    ✔ should initialize sucessfully
    reverts if init address = 0x0:
      ✔ acm
      ✔ treasury
      ✔ stableToken
    reverts if fee init value is invalid
      ✔ feeIn
      ✔ feeOut
  Admin functions
    pause()
      ✔ should revert if not authorised
      ✔ should pause if authorised
      ✔ should revert if already paused
    resume()
      ✔ should revert if not authorised
      ✔ should resume if authorised
      ✔ should revert if already resumed
    setFeeIn(uint256)
      ✔ should revert if not authorised
      ✔ should revert if fee is invalid
      ✔ set the correct fee
    setFeeOut(uint256)
      ✔ should revert if not authorised
      ✔ should revert if fee is invalid
      ✔ set the correct fee
    setVAIMintCap(uint256)
      ✔ should revert if not authorised
      ✔ should set the correct mint cap
    setVenusTreasury(uint256)
      ✔ should revert if not authorised
      ✔ should revert if zero address
      ✔ should set the treasury address
    setOracle(address)
      ✔ should revert if not authorised
      ✔ should revert if oracle address is zero
      ✔ should set the oracle (41ms)
  Pause logic
    ✔ should revert when paused and call swapVAIForStable(address,uint256)
    ✔ should revert when paused and call swapStableForVAI(address,uint256)
  Swap functions
    swapVAIForStable(address,uint256)
      ✔ should revert if receiver is zero address
      ✔ should revert if sender has insufficient VAI balance  (38ms)
      ✔ should revert if VAI transfer fails  (52ms)
      ✔ should revert if VAI to be burnt > vaiMinted
      should sucessfully perform the swap
        Fees: 10%
          ✔ stable token = 1$  (64ms)
          ✔ stable token < 1$  (59ms)
          ✔ stable token > 1$  (63ms)
        Fees: 0%
          ✔ stable token = 1$  (53ms)
          ✔ stable token < 1$  (55ms)
          ✔ stable token > 1$  (57ms)
    swapStableForVAI(address,uint256)
```

```
      ✔ should revert if receiver is zero address
      ✔ should revert if VAI mint cap will be reached  (54ms)
      should sucessfully perform the swap
        Fees: 10%
          ✔ stable token = 1$  (77ms)
          ✔ stable token > 1$  (78ms)
          ✔ stable token < 1$  (77ms)
        Fees: 0%
          ✔ stable token = 1$  (70ms)
          ✔ stable token > 1$  (70ms)
          ✔ stable token < 1$  (70ms)
PSM: 6 decimals
  initialization
    ✔ should revert if contract already deployed
    ✔ should initialize sucessfully
    reverts if init address = 0x0:
      ✔ acm
      ✔ treasury
      ✔ stableToken
    reverts if fee init value is invalid
      ✔ feeIn
      ✔ feeOut
  Admin functions
    pause()
      ✔ should revert if not authorised
      ✔ should pause if authorised
      ✔ should revert if already paused
    resume()
      ✔ should revert if not authorised
      ✔ should resume if authorised
      ✔ should revert if already resumed
    setFeeIn(uint256)
      ✔ should revert if not authorised
      ✔ should revert if fee is invalid
      ✔ set the correct fee
    setFeeOut(uint256)
      ✔ should revert if not authorised
      ✔ should revert if fee is invalid
      ✔ set the correct fee
    setVAIMintCap(uint256)
      ✔ should revert if not authorised (38ms)
      ✔ should set the correct mint cap
    setVenusTreasury(uint256)
      ✔ should revert if not authorised (44ms)
      ✔ should revert if zero address
      ✔ should set the treasury address
    setOracle(address)
      ✔ should revert if not authorised
      ✔ should revert if oracle address is zero
      ✔ should set the oracle (53ms)
  Pause logic
    ✔ should revert when paused and call swapVAIForStable(address,uint256)
    ✔ should revert when paused and call swapStableForVAI(address,uint256)
  Swap functions
    swapVAIForStable(address,uint256)
      ✔ should revert if receiver is zero address
      ✔ should revert if sender has insufficient VAI balance  (53ms)
      ✔ should revert if VAI transfer fails  (65ms)
      ✔ should revert if VAI to be burnt > vaiMinted  (53ms)
      should sucessfully perform the swap
        Fees: 10%
          ✔ stable token = 1$  (92ms)
          ✔ stable token < 1$  (95ms)
          ✔ stable token > 1$  (95ms)
        Fees: 0%
          ✔ stable token = 1$  (84ms)
          ✔ stable token < 1$  (81ms)
          ✔ stable token > 1$  (82ms)
    swapStableForVAI(address,uint256)
      ✔ should revert if receiver is zero address
      ✔ should revert if VAI mint cap will be reached  (81ms)
      should sucessfully perform the swap
```

```
            Fees: 10%
              ✔ stable token = 1$  (108ms)
              ✔ stable token > 1$  (109ms)
              ✔ stable token < 1$  (121ms)
            Fees: 0%
              ✔ stable token = 1$  (100ms)
              ✔ stable token > 1$  (101ms)
              ✔ stable token < 1$  (99ms)


  142 passing (33s)
```

Code Coverage

The code coverage was generated for contracts in contracts/PegStability using `npx hardhat coverage --testfiles "./tests/hardhat/VAI/PegStability.ts"` and .solcover.js provided below. We recommend adding more tests to ensure the branch coverage is larger than 90% before deploying the contracts.

`.solcover.js` file set to:

```
module.exports = {
    skipFiles: ['Comptroller', 'DelegateBorrowers', 'Governance', 'InterestRateModels', 'Lens',
'Liquidator', 'Oracle', 'Swap', 'test', 'Tokens', 'Utils', 'Vault', 'VRTVault', 'XVSVault'],
  };
```

We recommend adding tests that ensure the accuracy of the swap preview functions.

Update
Tests were slightly modified to confirm the functionality of the custom errors they are now using instead of revert statements.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
PegStability/	90	83.33	85	88.1	
IVAI.sol	100	100	100	100	
PegStability.sol	90	83.33	85	88.1	... 424,425,428
All files	90	83.33	85	88.1	

Changelog

- 2023-07-27 - Initial report
- 2023-08-04 - Updated the initial report to include fix review according to commit `ef56f38`

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

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 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.