Quantstamp

Venus - Leverage Strategies Manager

Executive Summary

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

Type	Leveraged Position Peripheral Contract
Timeline	2025-12-03 through 2025-12-07
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Audit Scope Venus Docs
Source Code	• https://github.com/VenusProtocol/venus-periphery #6a4ba3e
Auditors	• Cameron Biniamow Auditing Engineer • Ibrahim Abouzied Auditing Engineer • Rabib Islam Senior Auditing Engineer

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

Summary of Findings

The Venus Leverage Strategies Manager is a peripheral contract that enables users to enter and exit single- and multi-asset leveraged positions in Venus core pools. Users can enter leveraged positions through a variety of options, such as taking on borrowed debt via a flashloan and using the borrowed assets as collateral in the same Venus market, or swapping to another Venus market token before supplying it as collateral. Similarly, users can use flashloans to unwind their positions by borrowing assets to repay debt and redeem collateral.

The audit of the Venus Leverage Strategies Manager reveals three vulnerabilities and three auditor suggestions. Two low-severity issues were identified. VEN-1 describes the case where excess swapped assets are not returned to the user, and instead transferred to the Venus treasury. While unlikely, a user swapping too many assets would unfairly lose funds. VEN-2 points out the edge case where a user borrows more from a flashloan than is needed to repay their debt, and the excess flashloan funds are not included in the flashloan repayment. One information severity issue, VEN-3, notes that unaccrued interest is not updated before the initial account liquidation check, potentially preventing a user from entering a valid high-leverage position.

Three audit suggestions are listed for code optimizations and adherence to best practices. The recommendations include: simplifying the mint amount in the `_handleEnterSingleAsset()` function, which unnecessarily borrows funds for fee repayment after minting, generating excess gas usage; removing the unused `Ownable2StepUpgradeable` dependency, as it does not affect any function in this contract; and establishing a whitelist for market tokens to mitigate risks associated with low liquidity assets during swap actions.

Update: The Venus team fixed or acknowledged all vulnerabilities and suggestions. VEN-1, VEN-2, and VEN-3 were fixed as recommended. Suggestions S1-S3 were acknowledged with provided reasoning.

ID	DESCRIPTION	SEVERITY	STATUS
VEN-1	Excess Swapped Assets Not Returned to the User	• Low	Fixed
VEN-2	Collateral Exit Flashloan Repayment Does Not Consider Excess Borrow	• Low	Fixed
VEN-3	Accrued Interest Is Not Updated Before the Account Liquidation Check	• 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

Files Included

Repo: `https://github.com/VenusProtocol/venus-periphery`

Included Paths: `contracts/LeverageManager`

Operational Considerations

1. **Flashloan Enabled Markets**: Users can enter a leveraged position via a flashloan if the market has enabled flashloans. If flashloans are disabled for a market while a user has an open leveraged position, the user may not be able to exit the position without risking liquidation or utilizing flashloans from other protocols in combination with custom contract development.
2. **Swap Data**: Entering and exiting multi-asset leveraged positions requires signed swap calldata from the Venus team. Transactions containing invalid swap calldata will fail.
3. **Upgradeable Contracts**: The `LeverageStrategiesManager` is upgradeable. Any contract implementation upgrades should be audited to ensure the logic functions as intended and there are no storage collisions.
4. **Delegate Approval**: Users must set the `LeveragedStrategiesManager` contract as an approved delegate before entering or exiting leveraged positions.
5. **Flashloan Fee**: Using flashloans to enter or exit leveraged positions may accrue flashloan fees. Any accrued fees are repaid to the respective market through borrowed funds or by redeeming collateral.
6. **Liquidated Positions**: High-leverage positions are at an increased risk of liquidation, which can lead to a loss of user funds.
7. **Incompatibility with Native and Non-Standard ERC-20 Tokens**: Markets that have native or non-standard underlying tokens are incompatible with the `LeveragedStrategiesManager` contract. Fee-on-transfer tokens will cause transaction failures when entering markets and repaying flashloans. Additionally, the `LeveragedStrategiesManager` contract is not designed to handle markets with native underlying tokens.

Key Actors And Their Capabilities

- **Comptroller**: The Venus `Comptroller` contract handles the execution of flashloans requested by the `LeveragedStrategiesManager` contract. Flashloaned assets are transferred to the `LeveragedStrategiesManager` contract to enter or exit leveraged positions on behalf of the user. Used by the `LeveragedStrategiesManager` contract to query a user's borrowing power and if the user faces liquidation before and after updating a position.
- **Venus Swap API**: Generates a signed single-use swap payload used when users enter or exit applicable leveraged positions. The encoded data is executed by the out-of-scope `SwapHelper` contract to swap one asset for another, then return the swapped assets to the `LeveragedStrategiesManager` contract.
- **Swap Helper Contract**: A contract called by the `LeveragedStrategiesManager` contract, which handles the execution and routing of swaps using encoded calldata generated by the Swap API.

Findings

VEN-1 Excess Swapped Assets Not Returned to the User

• Low ⓘ Fixed

Update
The client fixed the issue in commit `d5dc0ac92d347eaaf9fd9d6f492aa697e8698f33`. Excess assets are returned to the initiator rather than to the treasury.

File(s) affected: `contracts/LeverageManager/LeverageStrategiesManager.sol`

Description: During the execution of `exitLeverage()`, a flash loan is initially used to repay the debt incurred by a multi-asset leveraged position. A user-specified amount of collateral is then redeemed and swapped to repay the flash loan. If the user specifies a collateral amount that is too large to swap, and the swapped amount exceeds the flashloan repayment amount, the excess funds are sent to the Venus treasury instead of the user. This behavior differs from other cases in which the user receives excess funds that may remain in the contract after a leveraged position is entered or exited.

In most cases, the user will likely specify only enough collateral to reasonably cover the flashloan repayment amount, limiting their losses to the treasury; however, if the user accidentally specifies too large a collateral amount to swap, they can experience significant losses.

Recommendation: Return excess borrowed assets to the user rather than to the treasury.

VEN-2 Collateral Exit Flashloan Repayment Does Not Consider Excess Borrow

• Low ⓘ Fixed

Update
The client fixed the issue in commit `ecf7bdce016b13dab996da453913fb23f71cf5f4`. The flashloan repayment check now uses the borrowed asset balance in the contract.

File(s) affected: `contracts/LeverageManager/LeverageStrategiesManager.sol`

Description: In the `_handleExitCollateral()` function, the flashloan amount is used to repay borrowed assets in a multi-asset leveraged position. Later in the function, a specified collateral amount is redeemed and then swapped to repay the flash loan. The following check ensures that the swap output amount is sufficient to cover the flashloan repayment:

```
if (swappedBorrowedAmountOut < flashLoanRepayAmount) {
    revert InsufficientFundsToRepayFlashloan();
}
```

However, the check does not consider the case where the flashloan amount exceeded the `borrowMarket.borrowBalanceCurrent(onBehalf)` amount, and excess assets from the flashloan remain in the contract.

While it is unlikely that a user would specify a flashloan amount that exceeds their borrow balance, if they do, they would unnecessarily need to swap more collateral than required to repay the flashloan. Additionally, with any excess borrowed assets being sent to the Venus treasury, the user would forefit all excess flashloan assets.

Recommendation: Use the borrowed asset balance in the `LeverageStrategiesManager` contract instead of `swappedBorrowAmountOut`, which would include any excess flashloan assets.

```
if (borrowedAsset.balanceOf(address(this)) < flashLoanRepayAmount) {
    revert InsufficientFundsToRepayFlashloan();
}
```

VEN-3
Accrued Interest Is Not Updated Before the Account Liquidation Check

● Informational ⓘ Fixed

Update
The client fixed the issue in commit `d6e1e95f247199489f35cc2621521874754eb683`. Interest is accrued for each market before the liquidation check is performed.

File(s) affected: `contracts/LeverageManager/LeverageStrategiesManager.sol`

Description: Before calling `_checkAccountSafe()` when entering or exiting a leveraged position, the market may have accrued unaccounted-for interest. Therefore, a user entering a high-leveraged position may fail the initial liquidation check even though it would pass if unaccrued interest were considered. Since interest accrues whenever a market borrows, mints, redeems, or repays, the liquidation check after the position update will use the latest accumulated interest amount.

Recommendation: Consider calling `accrueInterest()` for the collateral and borrow markets (if applicable) before performing the initial liquidation check. Otherwise, inform users that high-leveraged positions may fail unless `accrueInterest()` is manually executed before entering a position.

Auditor Suggestions

S1 Avoid a Borrow in `_handleEnterSingleAsset()`

Acknowledged

Update
The client acknowledged the suggestion and provided the following explanation:

```
This optimization would alter the economic outcome compared
  to other enter functions ( _handleEnterCollateral ,
  _handleEnterBorrow ). In single-asset leverage, the user would
  receive less collateral and less debt, whereas other functions
  mint the full flash loan amount and borrow fees separately.
  Maintaining consistent gross position behavior across all enter
  functions is preferred for predictability and UI calculation
  consistency.
```

File(s) affected: `contracts/LeverageManager/LeverageStrategiesManager.sol`

Description: The `_handleEnterSingleAsset()` function will mint the full token balance, only to later borrow the amount needed to repay the `collateralAmountFees`. Given that no swap takes place, the the code can be simplified by withholding the amount needed to repay the fees from the mint.

Recommendation: Subtract the fees from the mint amount, and remove the call to borrow and repay the funds. For example:

```
function _handleEnterSingleAsset(
    address onBehalf,
    IVToken market,
    uint256 flashloanedCollateralAmount,
    uint256 collateralAmountFees
) internal returns (uint256 flashLoanRepayAmount) {
    IERC20Upgradeable collateralAsset = IERC20Upgradeable(market.underlying());

    uint256 totalCollateralAmountToMint = flashloanedCollateralAmount + collateralAmount -
collateralAmountFees;
    collateralAsset.forceApprove(address(market), totalCollateralAmountToMint);

    uint256 err = market.mintBehalf(onBehalf, totalCollateralAmountToMint);
    if (err != SUCCESS) {
        revert MintBehalfFailed(err);
    }
    flashLoanRepayAmount = collateralAmountFees;
}
```

S2 Ownable Is Unused

Acknowledged

Update

The client acknowledged the suggestion and provided the following explanation:

```
Retained for forward compatibility. Future versions may introduce
  owner-restricted functions (configurable fees, emergency pause,
  swap helper management). Removing now would require
  storage-layout-breaking upgrade to reintroduce later.
```

File(s) affected: `contracts/LeverageManager/LeverageStrategiesManager.sol`

Description: The contract inherits from `Ownable2StepUpgradeable` but does not use the `onlyOwner` modifier.

Recommendation: Remove the unused dependency.

S3 Consider Adding Whitelist to Prevent Use of Low Liquidity Markets

Acknowledged

Update

The client acknowledged the suggestion and provided the following explanation:

```
Market whitelisting deferred to Comptroller's existing
  governance-controlled listing process. Users control slippage
  risk via  minAmountOutAfterSwap  . Low-liquidity markets self-limit
  through poor swap execution.
```

File(s) affected: `contracts/LeverageManager/LeverageStrategiesManager.sol`

Description: The `LeverageStrategiesManager` contract allows leveraging any market listed in the `Comptroller` and swapping through any route supported by the `SwapHelper`. However, this approach could expose users to assets with low liquidity, which may experience greater fluctuations in price during swaps and may therefore be difficult to exit out of.

Recommendation: Implement a whitelist for both collateralMarket and borrowedMarket tokens that are deemed safe for leveraged trading.

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 pose 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

Repo: `https://github.com/VenusProtocol/venus-periphery`

- `ef1...762 ./contracts/LeverageManager/ILeverageStrategiesManager.sol`
- `8f6...c1a ./contracts/LeverageManager/LeverageStrategiesManager.sol`

Test Suite Results

The test suite was run using `npx hardhat test "tests/hardhat/LeverageManager/LeverageStrategiesManager.ts"`. All 78 test cases passed and were run successfully.

Update: The client added 30 test cases with the fixes. All 108 test cases passed.

```
  LeverageStrategiesManager
    Deployment & Initialization
      ✔ should deploy successfully
      ✔ should deploy with correct immutable variables
      ✔ should revert on deployment when comptroller address is zero (123ms)
      ✔ should revert on deployment when protocolShareReserve address is zero (127ms)
      ✔ should revert on deployment when swapHelper address is zero (96ms)
      ✔ should initialize correctly
      ✔ should revert if initialized twice
    enterSingleAssetLeverage
      Validation
        ✔ should revert when flash loan amount is zero
        ✔ should revert when collateral market is not listed
        ✔ should revert when user has not set delegation
        ✔ should revert when user did not approve enough collateral for transfer (95ms)
        ✔ should revert with MintBehalfFailed when mint fails in enterSingleAssetLeverage (149ms)
      Success Cases
        ✔ should enter leveraged position with single collateral successfully without seed (315ms)
        ✔ should enter leveraged position with single collateral successfully with seed (304ms)
        ✔ should verify account is safe after entering leveraged position (250ms)
        ✔ should transfer dust to initiator after entering leveraged position (211ms)
        ✔ should succeed when user is already in the collateral market (233ms)
        ✔ should emit DustTransferred event when dust is returned to user (368ms)
        ✔ should handle zero fees flash loan successfully (402ms)
    enterLeverage
      Validation
        ✔ should revert when flash loan amount is zero
        ✔ should revert when collateral market is not listed
        ✔ should revert when borrow market is not listed
        ✔ should revert when user has not set delegation
        ✔ should revert when user did not approve enough collateral for transfer (101ms)
        ✔ should revert when swap fails (132ms)
        ✔ should revert when aftrer swap, received less collateral than minAmountCollateralAfterSwap
(134ms)
        ✔ should revert with MintBehalfFailed when mint fails in enterLeverage (177ms)
      Success Cases
        ✔ should enter leveraged position with collateral successfully (414ms)
        ✔ should transfer dust to initiator after entering leveraged position (309ms)
        ✔ should enter leveraged position with non-zero collateral seed successfully (389ms)
    enterLeverageFromBorrow
      Validation
        ✔ should revert when flash loan amount is zero
        ✔ should revert when collateral market is not listed
        ✔ should revert when borrow market is not listed
        ✔ should revert when user has not set delegation
        ✔ should revert when swap fails (110ms)
        ✔ should revert when aftrer swap, received less collateral than minAmountCollateralAfterSwap
(132ms)
        ✔ should fail when user did not approve enough borrowed tokens for transfer (108ms)
        ✔ should revert with MintBehalfFailed when mint fails in enterLeverageFromBorrow (201ms)
      Success Cases
        ✔ should enter leveraged position with borrowed successfully (304ms)
        ✔ should transfer dust to initiator after entering leveraged position (291ms)
        ✔ should enter leveraged position with non-zero borrowed seed successfully (341ms)
        ✔ should transfer borrow seed amount correctly when non-zero (294ms)
    exitLeverage
      Validation
        ✔ should revert when flash loan amount is zero
        ✔ should revert when collateral market is not listed
        ✔ should revert when borrow market is not listed
        ✔ should revert when user has not set delegation
        ✔ should revert when swap fails (475ms)
        ✔ should revert when after swap is received less borrowed than minAmountBorrowedRepayAfterSwap
(517ms)
        ✔ should revert with InsufficientFundsToRepayFlashloan when swap returns less than flash loan
repayment (576ms)
      Success Cases
        ✔ should exit leveraged position successfully (515ms)
        ✔ should emit DustTransferred events for collateral to user and borrow to treasury (575ms)
        ✔ should transfer collateral dust to initiator and borrow dust to treasury after exiting (546ms)
        ✔ should call protocolShareReserve.updateAssetsState when transferring dust to treasury (506ms)
        ✔ should handle flash loan amount exceeding actual debt in exitLeverage (518ms)
    exitSingleAssetLeverage
      Validation
        ✔ should revert when flash loan amount is zero
        ✔ should revert when collateral market is not listed
        ✔ should revert when user has not set delegation
      Success Cases
        ✔ should exit leveraged position with single collateral successfully (498ms)
        ✔ should verify account is safe after exiting leveraged position (408ms)
        ✔ should transfer dust to initiator after exiting leveraged position (558ms)
        ✔ should allow entering and exiting leveraged position multiple times (826ms)
      Error Cases
        ✔ should revert with BorrowBehalfFailed when borrow is not allowed on market (168ms)
        ✔ should revert when REPAY action is paused during exit (343ms)
        ✔ should revert when REDEEM action is paused during exit (409ms)
        ✔ should revert when BORROW action is paused during enter (184ms)
        ✔ should revert with InsufficientFundsToRepayFlashloan when redeemed amount is insufficient in
exitSingleAssetLeverage (452ms)
      Edge Cases
        ✔ should not emit DustTransferred when dust amount is zero after exit (439ms)
        ✔ should handle flash loan amount exceeding actual debt in exitSingleAssetLeverage (407ms)
        ✔ should handle re-entering same market position after exiting (951ms)
    executeOperation
      Access Control
        ✔ should revert when caller is not comptroller
        ✔ should revert when onBehalf is different than operation initiator
        ✔ should revert when caller is not comptroller
        ✔ should revert when vTokens, amounts and premiums length is not 1
        ✔ should revert when called not as a callback of a flash loan
    Multi-user scenarios
      Concurrent Operations
        ✔ should allow different users to perform leverage operations concurrently (789ms)
        ✔ should isolate transient storage between different user operations (748ms)
    EnterMarketFailed scenarios
      ✔ should succeed when user is already in the market via enterLeverage (272ms)
      ✔ should succeed when user is already in the market via enterLeverageFromBorrow (340ms)


  78 passing (36s)
```

Code Coverage

Test coverage was generated by running `npx hardhat coverage --testfiles "tests/hardhat/LeverageManager/LeverageStrategiesManager.ts"`. The in-scope files have moderate test coverage. The audit team recommends improving the test suite to acheive 100% coverage.

Update: The additional test cases added during the fixes slightly raised the branch coverage for the `LeverageStrategiesManager` contract. The audit team still recommends improving the test suite to exceed 90% branch coverage.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	100	100	100	100	
Interfaces.sol	100	100	100	100	
contracts/LeverageManager/	98.71	77.78	95.45	94.55	
ILeverageStrategiesManager.sol	100	100	100	100	
LeverageStrategiesManager.sol	98.71	77.78	95.45	94.55	... 582,690,695
contracts/PositionSwapper/	0	0	0	0	
ISwapHelper.sol	100	100	100	100	
PositionSwapper.sol	0	0	0	0	... 428,438,439
WBNBSwapHelper.sol	0	0	0	0	... 130,131,133
contracts/SwapHelper/	76.92	40	62.5	62.5	
SwapHelper.sol	76.92	40	62.5	62.5	... 221,224,225
All files	58.05	33.88	54.17	58.69	

Changelog

- 2025-12-08 - Initial report
- 2025-12-11 - Final report

About Quantstamp

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' 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 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 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 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 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