Quantstamp

Venus Income Allocation

Executive Summary

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

Type	DeFi income allocator
Timeline	2023-08-28 through 2023-09-19
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Audit scope Document Protocol Documentation Whitepaper
Source Code	• VenusProtocol/venus-protocol #2e0f5f6 • VenusProtocol/venus-protocol #b11d297 • VenusProtocol/isolated-pools #92353cf • VenusProtocol/protocol-reserve #dfb653d
Auditors	• Nikita Belenkov Auditing Engineer • Michael Boyle Auditing Engineer • Shih-Hung Wang Auditing Engineer • Mostafa Yassin Auditing Engineer

Documentation quality	High
Test quality	Medium
Total Findings	19 Fixed: 3 Acknowledged: 14 Mitigated: 2
High severity findings	0
Medium severity findings	2 Acknowledged: 2
Low severity findings	12 Fixed: 2 Acknowledged: 8 Mitigated: 2
Undetermined severity findings	0
Informational findings	5 Fixed: 1 Acknowledged: 4

Summary of Findings

This audit covered an upgrade to the Venus Protocol that is responsible for distributing the incomes generated by the protocol. Venus Protocol is a DeFi protocol deployed on the BNB chain, which combines stablecoin minting and algorithmic money markets.

To achieve this goal, changes had to be made across multiple contracts and introduce new accumulator contracts along with a control flow upgrade to an immutable contract. The following changes were covered in this audit:

1. Upgrades have been made to VToken contracts for the core and isolated pools, where the income from spread and liquidations is now sent to the Protocol Share Reserve contract.
2. The VBNBAdmin contract has been introduced to handle BNB income, wrap it, and send it to the Protocol Share Reserve contract.
3. The Protocol Share Reserve contract has been introduced to receive this income from multiple sources and distribute it based on certain upgradable ratios.

Overall, the code is well written and has good documentation. The test coverage varies along with the test suite between different parts of the project. The main concerns highlighted in this report are based on potential Denial of Service due to function invocation of arbitrary addresses, see VEN-2, VEN-5, and VEN-12; non-standard tokens leading to accountancy issues, see VEN-1 and other individual low concerns mainly across the newly developed contracts.

The audit team has strictly covered the files that are in the Scope section and any other files or system was not in the scope of this audit. It is highly recommended for the Venus team to address all the issues highlighted in this report.

Fix Review Update

Venus protocol team has either fixed or acknowledged all the issues highlighted in the report.

ID	DESCRIPTION	SEVERITY	STATUS
VEN-1	Non-Standard Tokens Will Cause the Accounting System to Break	● Medium ⓘ	Acknowledged
VEN-2	DoS of `_releaseFund()` if Incorrect Address Is Set	● Medium ⓘ	Acknowledged
VEN-3	Upgradability	● Low ⓘ	Acknowledged
VEN-4	Missing Input Validation	● Low ⓘ	Mitigated
VEN-5	DoS Possible if Incorrect `protocolShareReserve` Address Is Used	● Low ⓘ	Acknowledged
VEN-6	State Variables Not Set Upon Initialization	● Low ⓘ	Acknowledged
VEN-7	It Is Not Possible to Remove Items From the `distributionTargets` Array	● Low ⓘ	Fixed
VEN-8	One Can Add Zero Percentage Destinations to Waste Gas	● Low ⓘ	Acknowledged
VEN-9	Native Token Transfer Is Still Possible	● Low ⓘ	Acknowledged
VEN-10	Some Loops Do Not Use the `_ensureMaxLoops()` Check	● Low ⓘ	Acknowledged
VEN-11	ERC-20 Dust Left in the Contract After `_releaseFund()`	● Low ⓘ	Acknowledged
VEN-12	Potential DoS Issue in `ProtocolShareReserve`'s `addOrUpdateDistributionConfigs()`	● Low ⓘ	Mitigated
VEN-13	Unhandled Error Code from `_reduceReservesFresh()` in `VToken`	● Low ⓘ	Acknowledged
VEN-14	Inconsistency of Error Handling for vToken's Privileged Functions	● Low ⓘ	Fixed
VEN-15	Outdated Solidity Version	● Informational ⓘ	Acknowledged
VEN-16	Safe Math Function Not Used For Block Delta Calculation	● Informational ⓘ	Fixed
VEN-17	Inconsistent Access Control Between `vBNB` and Other Core Pool vTokens	● Informational ⓘ	Acknowledged
VEN-18	Potential Limitation in `VBNBAdmin` Design	● Informational ⓘ	Acknowledged
VEN-19	State Inaccuracies During External Calls in `ProtocolShareReserve.releaseFunds()`	● 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 audit scope included contracts across 4 code repositories. 2 repositories include an update to the `VToken` implementation, while the other 2 include new contracts used to manage the income separation and forwarding.

Files Included

1. Core pool: `VenusProtocol/venus-protocol @ 2e0f5f64f49aba3150e49c7a7e34a28826726f20`
   1. contracts/Tokens/VTokens/VToken.sol
   2. contracts/Tokens/VTokens/VTokenInterfaces.sol
   3. contracts/Utils/ErrorReporter.sol
2. BNB income: `VenusProtocol/venus-protocol @ b11d2972dbbf9855a7560f26745fae783bc15e7e`
   1. contracts/Admin/VBNBAdmin.sol
   2. contracts/Admin/VBNBAdminStorage.sol
3. Isolated pools: `VenusProtocol/isolated-pools @ 92353cf35ddbbfdd67ee255ac095ee861d6bb7fb`
   1. contracts/VToken.sol
   2. contracts/VTokenInterfaces.sol
4. ProtocolShareReserve: `VenusProtocol/protocol-reserve @ dfb653d2e3fe163a248bbd9f8951cd6b96b06390`
   1. contracts/ProtocolReserve/ProtocolShareReserve.sol.
   2. contracts/Interfaces/IIncomeDestination.sol
   3. contracts/Interfaces/IPrime.sol
   4. contracts/Interfaces/IProtocolShareReserve.sol
   5. contracts/Interfaces/IVToken.sol
   6. contracts/Interfaces/ComptrollerInterface.sol
   7. contracts/Interfaces/PoolRegistryInterface.sol

Findings

VEN-1

Non-Standard Tokens Will Cause the Accounting System to Break

● Medium ⓘ Acknowledged

Update

The client has acknowledged the issue and commented: "We will prevent adding such non-standard ERC20 tokens that decrease the balance of an account. We have an internal guideline for new markets. Rebase tokens, or fee-on-transfer tokens, for example, are not initially supported by the protocol".

File(s) affected: `contracts/ProtocolReserve/ProtocolShareReserve.sol`

Description: `updateAssetsState()` function is usually called when a transfer of ERC20 assets has been made to the contract. It can also be called by any user of the platform. The idea behind this function is to account for newly transferred ERC20 tokens by taking the difference between the current balance of the contract and the accounted balance of the contract. It is assumed that there should be no case where the `currentBalance` will be less than the already accounted `assetReserve`.

This is not strictly the case, as if the asset is a rebasing token such as Ampelforth, the number of tokens in the contract balance will vary depending on the current market conditions and the underlying peg-keeping algorithm. These types of balance changes are also common in other types of non-standard tokens.

When `releaseFunds()` is called with the correct parameters, the income of such rebasing token would be distributed based on the last `updateAssetsState()` balance, which might not be the same as the current contract balance and could be potentially lower than the `schemaBalances` accounted balance. Leading to an issue where the last transfer will fail due to insufficient funds in the contract.

Recommendation: `updateAssetsState()` should also account for the decrease in contract balance for such tokens when called.

VEN-2 DoS of `_releaseFund()` if Incorrect Address Is Set

● Medium ⓘ Acknowledged

Update
The client has acknowledged the issue and commented: "We will be deploying and configuring the destination contracts through Governance process therefore we will beforehand verify that the destinations support the correct interface. Moreover, the Venus Treasury (0xf322942f644a996a617bd29c16bd7d231d9f35e9) is one of the destinations we'll configure. It's not upgradable, and it will accept the execution of the updateAssetsState() function via its fallback function. Not sure if we can implement something similar to the suggested mitigation".

File(s) affected: `contracts/ProtocolReserve/ProtocolShareReserve.sol`

Description: During the `_releaseFund()` function call, `DistributionConfig` s are iterated over, and the funds are divided between the provided addresses via an ERC20 transfer. After the ERC20 token transfer, the `updateAssetsState()` method of `IIncomeDestination` is invoked, so that the tokens are accounted for correctly. The issue arises from the fact that the method is called on an arbitrary contract at the address `_config.destination`, which is added to the array. Therefore, if an address is added that does not have the `updateAssetsState()` method implemented, this call will revert; hence, every call to `_releaseFund()` will revert, and the contract will not be usable, as it is not possible to remove items from the distribution array.

Recommendation: During the `addOrUpdateDistributionConfigs()` function, when a new address is set, make sure to call a predefined method on that address to make sure it is the correct contract, to avoid the issue described. Similar to `address.isIncomeDestination()`

VEN-3 Upgradability

● Low ⓘ Acknowledged

Update
The client has acknowledged the issue and commented: "Only the Normal timelock (0×939bD8d64c0A9583A7Dcea9933f7b21697ab6396) will be authorized to upgrade the implementations. So, they will be done by Governance".

File(s) affected: `VBNBAdmin.sol`, `contracts/Tokens/VTokens/VToken.sol`, `contracts/VToken.sol`, `ProtocolShareReserve.sol`

Description: Some of the contracts audited are upgradeable, and their functionality may be changed by privileged roles in the future.

Recommendation: Consider using a timelock contract for any upgrades. Make this clear to users via documentation.

VEN-4 Missing Input Validation

● Low ⓘ Mitigated

Update
Only recomendation for `setReduceReservesBlockDelta()` has been added and all the other recomendations have been acknowledged. Mitigated in commit `7bafbaa3be3ad904ac52a102b42426e9c6a56bb2` and `7f303f505d8278b846565485e4668e41f7206032`.

File(s) affected: `contracts/Tokens/VTokens/VToken.sol`, `contracts/VToken.sol`, `ProtocolShareReserve.sol`

Related Issue(s): SWC-123

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

The following functions are missing validation checks:

1. `ProtocolShareReserve`
   1. `initialize()`
      1. `_accessControlManager` should not be `address(0)`.
      2. `_loopsLimit` should not be `0`.
   2. `addOrUpdateDistributionConfigs()`
      1. `configs` should have a length greater than `0`.
   3. `releaseFunds()`
      1. `assets` should have a length greater than `0`.
2. `Isolated Pool VToken`
   1. `setReduceReservesBlockDelta()`
      1. `_newReduceReservesBlockDelta` should be greater than `0`. Otherwise, this would lead to an inconsistent reduction of reserves.
   2. `mint()`
      1. Allows to mint based on 0 underlying tokens
3. `Core Pool VToken`
   1. `setReduceReservesBlockDelta()`
      1. `_newReduceReservesBlockDelta` should be greater than `0`.

Recommendation: We recommend adding the relevant checks.

VEN-5
DoS Possible if Incorrect `protocolShareReserve` Address Is Used

• Low ⓘ Acknowledged

Update

The client has acknowledged the issue and commented: "We will set the ProtocolShareReserve contract through Governance process therefore we will verify that the contract supports the correct interface".

File(s) affected: `contracts/Tokens/VTokens/VToken.sol`, `contracts/Vtoken.sol`, `contracts/Admin/VBNBAdmin.sol`

Description: The contract has a function with `setProtocolShareReserve()` functionality that allows to set a new address for the `protocolShareReserve` contract. There are similar implementations of setters in the contract, but the main difference is that the new address is verified by a call to a `is<ExpectedContract>()` call, where Expected Contract is, for example, Comptroller. This is not the case for the `protocolShareReserve` contract.

Hence, it is possible for the new address not to be the expected contract and not to have an `updateAssetsState()` method implemented. This would lead to `_reduceReservesFresh()` failing on every call in the `VToken` context and hence the DoS of the usability of the contract, as the method would not exist and revert. In the context of `vBNBAdmin`, the function failing would be `reduceReserves()`.

This issue affects both the `VToken` contract implementations and the `vBNBAdmin` contract.

Recommendation: Add a `address.isProtocolShareReserve()` call to `setProtocolShareReserve()`.

VEN-6 State Variables Not Set Upon Initialization

• Low ⓘ Acknowledged

Update

The client has acknowledged the issue and commented: "New state variables that are introduced in this upgrade will be configured via VIP: https://github.com/VenusProtocol/vips/pull/67".

File(s) affected: `contracts/Tokens/VTokens/VToken.sol`, `contracts/VToken.sol`

Description: In both `VToken` contracts, the values for `reduceReservesBlockDelta` and `protocolShareReserve` are not set during initialization. This could cause unintended behavior at the time of deployment.

Recommendation: Consider initializing the variables in the `initialize()` function or update the variables within the same transaction as the upgrade.

VEN-7
It Is Not Possible to Remove Items From the `distributionTargets` Array

• Low ⓘ Fixed

Update

The `removeDistributionConfig()` function has been added that allows the removal of configs from the array. Fixed in commit c7ef4b0939446c84312b1e44b8e6034ac584ba8a.

File(s) affected: `contracts/ProtocolReserve/ProtocolShareReserve.sol`

Description: Currently, the distribution configs are updated via the `addOrUpdateDistributionConfigs()` function, and hence, the `distributionTargets` array is updated. It is currently possible to add an entry or update a percentage of existing entries but not remove entries from the array, hence leading to an issue where if an entry was added incorrectly, it could not be removed and only be set to 0 percentage, hence leading to be not accounted for, but would waste gas.

Recommendation: Add a feature that would allow the removal of an entry from the `distributionTargets` array.

VEN-8 One Can Add Zero Percentage Destinations to Waste Gas

• Low ⓘ Acknowledged

Update
The client has acknowledged the issue and commented: "We need to support a percentage of 0, to be sure the total percentage is 100 always after editing the rules. Moreover, with VEN-7, we could delete the unused rule in the same transaction, after editing the rules".

File(s) affected: `contracts/ProtocolReserve/ProtocolShareReserve.sol`

Description: During the `addOrUpdateDistributionConfigs()` function, it is possible to add a distribution config with a zero percentage. This would mean that no funds would be sent to that address, but the address would still be iterated over, hence wasting gas.

Recommendation: Consider if this is expected behavior, and if undesired, do not allow 0 percentage distribution configs.

VEN-9 Native Token Transfer Is Still Possible

• Low ⓘ Acknowledged

Update
The client has acknowledged the issue and commented: "We can later upgrade the contract to sweep native tokens if required".

File(s) affected: `contracts/ProtocolReserve/ProtocolShareReserve.sol`, `contracts/Admin/VBNBAdmin.sol`

Description: The idea of the `ProtocolShareReserve` contract is to accumulate income from ERC20 tokens and distribute it in the expected patterns between different addresses. It is not expected for the contract to receive native tokens, hence the absence of `payable` functions or the `receive()` method.

The EVM currently still supports the `SELFDESTRUCT` opcode. It is used to terminate a contract, remove the bytecode from the blockchain, and send any contract funds to a specified address. This transfer cannot be rejected, so if another contract self-destructs and points to `ProtocolShareReserve` as the recipient, then the contract will be funded from an unexpected source.

This issue is also applicable to the `vBNBAdmin` contract, as the specifications of the contract state that BNB tokens should only be received from the associated `vBNB` contract. Currently, the `receive()` function is protected with a required statement that only accepts funds from the `vBNB` contract. But with the issue described above, it is still possible to receive native tokens not from the expected source.

Recommendation: It is not possible to reject these transfers, but they should be taken into account if any unexpected behavior is caused.

VEN-10 Some Loops Do Not Use the `_ensureMaxLoops()` Check

• Low ⓘ Acknowledged

Update
The client has acknowledged the issue and commented: "We are using _ensureMaxLoops in addOrUpdateDistributionConfigs. This validation is enough as the distribution targets are used in all other loops".

File(s) affected: `contracts/ProtocolReserve/ProtocolShareReserve.sol`

Description: The `_ensureMaxLoops()` check is only used once in `addOrUpdateDistributionConfigs()` to limit the number of iterations through the distribution targets. It is not used to check other loops in the contract, such as the number of configs being added or updated, the number of assets used in `releaseFunds()`, the number of markets in `_accrueAndReleaseFundsToPrime()` and `_accruePrimeInterest()`, and the number of schemas or distribution targets in `_releaseFund()`.

The absence of these checks could lead to a potential gas issue and the transaction failing.

Recommendation: Determine if this is the intended design. If it is not, add the check where necessary.

VEN-11 ERC-20 Dust Left in the Contract After `_releaseFund()`

• Low ⓘ Acknowledged

Update
The client has acknowledged the issue and commented: "The round error is not significant".

File(s) affected: `contracts/ProtocolReserve/ProtocolShareReserve.sol`

Description: During the `_releaseFund()` call, the correct distribution patterns are applied to the collected ERC20 Income. The income is divided as a percentage between previously allocated addresses. The `transferAmount` is calculated via the following formula:

```
uint256 transferAmount = (schemaBalances[uint256(_config.schema)] * _config.percentage) / MAX_PERCENT
```

The division operator could lead to rounding errors, and hence, it is possible that the total transfer amount might not add up to the exactly expected `totalBalance`.

These leftover ERC20 tokens will be returned to the `assetsReserves` and reallocated at the next round, but potentially under a different allocation scheme. Hence, the users entitled to them might not receive them in the next allocation.

Recommendation: Consider if this is expected behavior. Otherwise, make sure the whole amount is redistributed.

VEN-12 Potential DoS Issue in `ProtocolShareReserve`'s `addOrUpdateDistributionConfigs()`

• Low Mitigated

Update
The issue has been partially mitigated by the added logic in `_releaseFund()` to skip token transfers and `updateAssetsState()` calls when the `transferAmount` is `0`. Mitigated in commit `65c4f91f239a454f6bb820242b02dcd64511901b`.

File(s) affected: `contracts/ProtocolReserve/ProtocolShareReserve.sol`

Description: The `ProtocolShareReserve.addOrUpdateDistributionConfigs()` function allows privileged roles to add or update the distribution configurations. Before these configurations are updated, the `_accrueAndReleaseFundsToPrime()` function is invoked to signal `Prime` to accrue interests. Then, `ProtocolShareReserve` transfers funds to `Prime` by invoking the `_releaseFund()` internal function with every asset in the `Prime` contract's market as an argument. According to the code comments, the accrual and release of funds to `Prime` are necessary before updating the distribution configurations.

However, this design introduces a potential issue where failing to accrue and release funds to `Prime` will block the configuration updates. Additionally, during the `_releaseFund()` function call, two external calls occur:

1. Tokens are transferred from `ProtocolShareReserve` to each configured destination.
2. The `updateAssetsState()` function is called on each destination contract.

If any of the external calls fails, the entire transaction fails. Failures can occur due to various reasons, including but not limited to:

1. A bug in the `Prime` contract causing the interest accrual call to fail.
2. A bug on the destination contracts causing the `updateAssetsState()` call to fail.
3. Token transfer failures to the destination contracts, which can occur due to token pausing, reverting on zero-value transfers, blacklisting of the sender or receiver, etc.

In cases 1 or 2, the `Prime` or destination contracts may require fixes and/or upgrades to unblock the `addOrUpdateDistributionConfigs()` call. In case 3, a potential solution could be the `Prime` contract removing the problematic tokens from its market. However, regardless of the specific issue, the problematic configuration can block the updates of others until the issue is resolved.

Recommendation: Consider evaluating whether the temporary blockage of updating distribution configurations might expose users to any potential risks. If so, consider allowing the privileged roles to decide which assets to release to `Prime`, following a similar approach in the `releaseFunds()` function, ensuring one type of asset does not impact the others.

Additionally, consider adding a special logic in `_releaseFund()` to skip token transfers and `updateAssetsState()` calls when the `transferAmount` is `0`. This will help prevent interactions with problematic or deprecated contracts after their configured `percentage` has been set to `0`.

Note that the issue described in case 3 may exist on one chain but not on another, as the token implementations may differ across different chains. If the contracts are intended to be deployed on several EVM chains in the future, these risks should be carefully considered and addressed.

VEN-13 Unhandled Error Code from `_reduceReservesFresh()` in `VToken`

• Low Acknowledged

Update
The client has acknowledged the issue and commented: "The protocol do the "best effort" to reduce reserves automatically, but there are scenarios (i.e. when there is not enough liquidity in the pool) where it won't be possible. The protocol shouldn't stop working on those cases, and it shouldn't be considered an error".

Description: The core pool's `VToken._reduceReservesFresh()` function can return a non-zero error code to indicate any failures during its execution. However, in the `accrueInterest()` function, when `_reduceReservesFresh()` is called, the returned error code is unhandled. Therefore, even though an error occurs during the execution of `reduceReservesFresh()` , the `accrueInterest()` function continues to execute, returning a `uint(Error.NO_ERROR)` to indicate that no errors have occurred. External contracts may incorrectly assume that `_reduceReservesFresh()` has executed successfully based on the `Error.NO_ERROR` return value.

Recommendation: If the execution of `_reduceReservesFresh()` should not fail during the `accrueInterest()` call, consider adding a check to ensure the return value of `_reduceReservesFresh()` is `Error.NO_ERROR` . This will help ensure that any errors in `_reduceReservesFresh()` are correctly handled for external contracts calling the `accrueInterest()` function.

VEN-14
Inconsistency of Error Handling for vToken's Privileged Functions

● Low ⓘ Fixed

Update

The original issue is fixed. Additionally, the code has been refactored, where now, the transaction will revert if the caller is unauthorized. For math errors or zero address inputs, the transaction reverts as well, ensuring consistency in error handling in the code. Fixed in commit `397e27202f006cd2f7c60ff7e15a39743e9398ab` .

File(s) affected: `contracts/Tokens/VTokens/VToken.sol`

Description: In the `VToken.setAccessControlManager()` function of the core pool, the caller is checked against the `admin` . If the caller is not authorized, the function returns an error code without reverting the transaction. However, the function comments state:

> @return uint 0=success, otherwise will revert

This inconsistency between the function's behavior and comments can confuse developers. Relying solely on the function's comments may lead developers to assume that a transaction will revert upon failure, while it may silently fail if `setAccessControlManager()` is called by an unauthorized caller.

Additionally, other privileged setters within the codebase also have inconsistent behavior. For example, `setProtocolShareReserve()` silently fails with an error code returned if the caller is unauthorized, while `setReduceReservesBlockDelta()` reverts the transaction during an unauthorized call.

Recommendation: To ensure clarity and code consistency, consider aligning the function's behavior with its comments. Since there are currently two different approaches to error handling within the codebase (i.e., returning an error code or reverting the transaction), consider documenting the error-handling approach used in each function within the function comments. This will help developers understand how errors are handled in each function for future development on the codebase.

VEN-15 Outdated Solidity Version

● Informational ⓘ Acknowledged

Update

The client has acknowledged the issue and commented: "The imported contracts are using 0.8.13, therefore it will require changes to other contracts in other repos to use 0.8.18. We prefer to stay as we are".

File(s) affected: `contracts/Admin/VBNBAdmin.sol` , `contracts/ProtocolReserve/ProtocolShareReserve.sol`

Description: These contracts are using solidity version `0.8.13` . While no high-severity issues are present in that version, it is currently recommended to use version `0.8.18` .

Exploit Scenario: Consider upgrading to the recommended solidity version of `0.8.18` .

VEN-16
Safe Math Function Not Used For Block Delta Calculation

● Informational ⓘ Fixed

Update

`subUInt()` has been added as recomended. Fixed in commit `51bf6fd75cc337d9c5716c9af4bbf667ca7914c0`

File(s) affected: `contracts/Tokens/VTokens/VToken.sol`

Description: The `accrueInterest()` function in the Core Pool `VToken` contract uses a library to perform arithmetic calculations due to the older Solidity version. The new implementation of this function, which adds the ability to call `_reduceReservesFresh()` if enough blocks have passed since it was last called, performs a calculation without using the library. While this should not be an issue, as the block number is steadily increasing, it is recommended to match previous standards.

Recommendation: Consider using a call to `subUInt()` when calculating the second instance of the block delta.

VEN-17 Inconsistent Access Control Between `vBNB` and Other Core Pool vTokens

● Informational ⓘ Acknowledged

Update

The client has acknowledged the issue and commented: "This behavior is intended. We want to allow anyone to invoke the reduceReserves functions in the vBNB contract, to pull the reserves from the market and send them to the ProtocolShareReserve, in a permissionless fashion. We don't need to do something similar in the VToken contract because reserves will be pushed from those tokens to the ProtocolShareReserve contract automatically (we didn't have that option for vBNB because that contract is not upgradable). If we want to disable in the future the automatic income allocation in one VToken (for example setting a huge value to the variable reduceReservesBlockDelta), we prefer to keep the reduceReserve function privileged".

File(s) affected: `contracts/Tokens/VTokens/VToken.sol`, `contracts/Admin/VBNBAdmin.sol`

Description: For the core pool, access to the `vToken._reduceReserves()` function is controlled by the `AccessControlManager`, which checks whether the caller is authorized to invoke this function. However, in the case of `vBNB`, the `vBNBAdmin.reduceReserves()` function operates without access control. This means that anyone can call the `reduceReserves()` function, which triggers the `vBNBAdmin` contract to call `vBNB._reduceReserves()`. This inconsistency between `vBNB` and the other core pool vTokens could potentially lead to different behaviors in their operations.

Recommendation: Confirm whether this design inconsistency between `vBNB` and other core pool vTokens is intentional. If it is not intended, consider implementing access controls for `vBNBAdmin.reduceReserves()` to align the behavior of `vBNB` with the other core pool vTokens.

VEN-18 Potential Limitation in `VBNBAdmin` Design

● Informational ⓘ Acknowledged

Update

The client has acknowledged the issue and commented: "The vBNBAdmin has no such scenarios where it needs to receive BNB that doesn't need to go to the ProtocolShareReserve contract. The fallback can only be triggered via owner and owner is timelock there (this function is called via Governance process always)".

File(s) affected: `contracts/Admin/VBNBAdmin.sol`

Description: The `VBNBAdmin` contract implements a `fallback()` function that is only callable by the admin, and this function forwards the call to the `vBNB` contract. This design allows the admin to call any function on the `vBNB` contract as needed.

However, if the call to the `vBNB` contract results in a BNB transfer back to the `VBNBAdmin` contract, no mechanism is currently in place to retrieve the received BNB specifically. These BNB will be considered token reserves and forwarded to the `ProtocolShareReserve` contract when the `reduceReserves()` is called.

Recommendation: Confirm whether this is an intended design. If so, consider updating the internal documentation to clarify the limitations and potential consequences of using the `VBNBAdmin` contract. If it is not intentional and there is a need to retrieve the received BNB separately, consider implementing the necessary functionality to achieve this.

VEN-19 State Inaccuracies During External Calls in `ProtocolShareReserve.releaseFunds()`

● Informational ⓘ Acknowledged

Update

The client has acknowledged the issue and commented: "Right now destination contracts don't read the state of the ProtocolShareReserve in the updateAssetState function".

File(s) affected: `contracts/ProtocolReserve/ProtocolShareReserve.sol`

Description: In the `ProtocolShareReserve._releaseFund()` function, funds are first transferred to destinations, followed by the call of the `updateAssetsState()` function on the destination contracts. Then, the `assetsReserves` and `totalAssetReserve` state variables are updated.

Note that since the state changes are applied after the external calls, the read of the contract state during the external call will be inaccurate.

It is important to note that the state changes are applied after the external calls. This means that during the external call of the `_releaseFund()` function if any destination contract (or a contract subsequently called by the destination contract during `updateAssetsState()`) reads the `assetsReserves` or `totalAssetReserve` state variables or invokes the `getUnreleasedFunds()` function, it will receive inaccurate values. These inaccuracies can potentially lead to accounting issues within the calling contracts.

Recommendation: Consider carefully examining the external calls made during `_releaseFund()` and assessing whether they read the state of `ProtocolShareReserve`. An alternative approach would be to follow the CEI pattern of updating the state before external calls, ensuring that the state values are accurate during these calls.

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

Code comments could be added to the following functions, structures, and interfaces:

1. Acknowledged BNB income: `VBNBAdminStorage.VTokenInterface`, `VBNBAdminStorage.IWBNB`, `VBNBAdminStorage.IProtocolShareReserve`
2. Acknowledged Isolated pools: `VTokenInterface.NewReduceReservesBlockDelta()`.
3. Acknowledged Protocol share reserve repo: `IPrime`, `IIncomeDestination`, `IProtocolShareReserve.updateAssetsState()`, `IVToken.underlying()`, `ProtocolShareReserves.DistributionConfig`

Adherence to Best Practices

1. Fixed `MAX_PERCENT` and percentage representation don't need to be a `uint256` as it should only be storing values up to 100.
2. Acknowledged The function `addOrUpdateDistributionConfigs()` accepts an array of `DistributionConfig` and adds them to the distribution target array. However, if the `configs` parameter contains duplicates, these will overwrite earlier entries. Consider checking off-chain that no duplicates exist.
3. Fixed In `ProtocolShareReserve`, using `calldata` instead of `memory` for parameters that are not modified will save gas.
4. Acknowledged In `ProtocolShareReserve` on line 244, the `_config` variable can be declared as `memory` because it is not being modified.
5. Fixed In `ProtocolShareReserve`, the `getSchema()` function is internal, but it is not named to indicate such. Consider prefixing the name with "_".
6. Acknowledged In `ProtocolShareReserve`, many of the internal functions are only used once. Moving the logic into the public function will reduce gas costs and may aid with readability.
7. Fixed In `ProtocolShareReserve`, there is one instance of a `require` statement being used instead of custom errors on line 176. Consider using the `InvalidAddress()` custom error.
8. Acknowledged Index up to 3 values in emitted events to save gas and make searching for them easier.
9. Acknowledged Cache array lengths in memory outside of a `for` loop to save gas.
10. Fixed In `Isolated Pool VToken`, consider combining lines 1525 and 1526 into a single statement.
11. Acknowledged Several functions in the Core Pool `VToken` have a name prefixed with "_" despite being public state-changing functions. Consider reserving this naming scheme for internal functions.
12. Fixed In the Core Pool `VToken` on line 1523, `totalReservesNew` is declared but not initialized. Consider removing this line and declaring it on line 1544, where it is assigned a value.
13. Acknowledged The following functions are not used:
    - `getCash()` in Core Pool `VToken`
14. Acknowledged Inconsistent usage of `onlyOwner` and `msg.sender == owner()`
15. Acknowledged `VBNBAdmin` uses a `fallback` function that accepts a payload and invokes it on the `vBNB` contract. Invoking this flow does not offer the best `UX`. A privileged function can be used instead.
16. Acknowledged Consider removing the unused error code: `SET_REDUCE_RESERVES_BLOCK_DELTA_OWNER_CHECK` in the core pool's `ErrorReporter.sol`.

17. Acknowledged Consider modifying the `vBNBAdmin.ReservesReduced()` event to align with the `ReservesReduced()` event defined in other core pool vTokens.

Adherence to Specification

Acknowledged The NatSpec comments for `setProtocolShareReserve()` were copied from `setReduceReservesBlockDelta()` and unchanged.

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

3159 result(s) found across the Core pool, BNB income, Isolated pools, and ProtocolShareReserve. Non-false positive findings have been included in the report.

Test Suite Results

ProtocolShareReserve and vBNBAdmin have tests covering the desired operation. The core pool and Isolated Pool test suites have been updated to include Protocol Share Reserve tests.

```
 1. Core pool

Comptroller
    _initializeMarket
The Hardhat Network tracing engine could not be initialized. Run Hardhat with --verbose to learn more.
The Hardhat Network tracing engine could not be initialized. Run Hardhat with --verbose to learn more.
      ✔ Supply and borrow state after initializing the market in the pool
    _setVenusSpeeds
      ✔ Revert on invalid supplySpeeds input
      ✔ Revert on invalid borrowSpeeds input
      ✔ Revert for unlisted market
      ✔ Revert on invalid borrowSpeeds input
      ✔ Updating non-zero speeds after setting it zero (67ms)

  Comptroller
    _setAccessControlManager
      ✔ Reverts if called by non-admin
      ✔ Reverts if ACM is zero address
      ✔ Sets ACM address in storage
    Access Control
      setCollateralFactor
        ✔ Should have AccessControl
      setLiquidationIncentive
        ✔ Should have AccessControl
      setMarketBorrowCaps
        ✔ Should have AccessControl
      setMarketSupplyCaps
        ✔ Should have AccessControl
      setProtocolPaused
        ✔ Should have AccessControl
      setActionsPaused
```

```
        ✔ Should have AccessControl
      supportMarket
        ✔ Should have AccessControl

  assetListTest
    enterMarkets
D
      ✔ properly emits events (104ms)
      ✔ adds to the asset list only once (374ms)
      ✔ the market must be listed for add to succeed (94ms)
      ✔ returns a list of codes mapping to user's ultimate membership in given addresses (70ms)
    exitMarket
      ✔ doesn't let you exit if you have a borrow balance (108ms)
      ✔ rejects unless redeem allowed (208ms)
      ✔ accepts when you're not in the market already (94ms)
      ✔ properly removes when there's only one asset (174ms)
      ✔ properly removes when there's only two assets, removing the first (205ms)
      ✔ properly removes when there's only two assets, removing the second (203ms)
      ✔ properly removes when there's only three assets, removing the first (219ms)
      ✔ properly removes when there's only three assets, removing the second (215ms)
      ✔ properly removes when there's only three assets, removing the third (261ms)
    entering from borrowAllowed
      ✔ enters when called by a vtoken (80ms)
      ✔ reverts when called by not a vtoken
      ✔ adds to the asset list only once (151ms)

  Comptroller
    constructor
      ✔ on success it sets admin to creator and pendingAdmin is unset (1182ms)
    _setLiquidationIncentive
      ✔ fails if incentive is less than 1e18 (47ms)
      ✔ accepts a valid incentive and emits a NewLiquidationIncentive event
    Non zero address check
      ✔ _setPriceOracle
      ✔ _setCollateralFactor
      ✔ _setPauseGuardian
      ✔ _setVAIController
      ✔ _setTreasuryData
      ✔ _setComptrollerLens
      ✔ _setVAIVaultInfo
      ✔ _setVenusSpeeds
    _setPriceOracle
      ✔ fails if called by non-admin
      ✔ accepts a valid price oracle and emits a NewPriceOracle event
    _setComptrollerLens
      ✔ fails if not called by admin
      ✔ should fire an event
    _setCloseFactor
      ✔ fails if not called by admin
    _setCollateralFactor
      ✔ fails if asset is not listed
      ✔ fails if factor is set without an underlying price
      ✔ succeeds and sets market (38ms)
    _supportMarket
      ✔ fails if asset is not a VToken
      ✔ succeeds and sets market
      ✔ cannot list a market a second time (38ms)
      ✔ can list two different markets (44ms)
    Hooks
      mintAllowed
        ✔ allows minting if cap is not reached
        ✔ reverts if supply cap reached
        ✔ reverts if market is not listed
      redeemVerify
        ✔ should allow you to redeem 0 underlying for 0 tokens
        ✔ should allow you to redeem 5 underlyig for 5 tokens
        ✔ should not allow you to redeem 5 underlying for 0 tokens

  Comptroller
    liquidateCalculateAmountSeize
      ✔ fails if borrowed asset price is 0
      ✔ fails if collateral asset price is 0
      ✔ fails if the repayAmount causes overflow  (46ms)
      ✔ fails if the borrowed asset price causes overflow  (41ms)
      ✔ reverts if it fails to calculate the exchange rate
      ✔ returns the correct value for
1000000000000000000,1000000000000000000,1000000000000000000,1000000000000000000,1000000000000000000
(54ms)
      ✔ returns the correct value for
2000000000000000000,1000000000000000000,1000000000000000000,1000000000000000000,1000000000000000000
(56ms)
      ✔ returns the correct value for
2000000000000000000,2000000000000000000,1420000000000000000,1300000000000000000,2450000000000000000
(55ms)
      ✔ returns the correct value for
2789000000000000000,5230480842000000000,77132000000000000000,1300000000000000000,1.000245e+22 (52ms)
      ✔ returns the correct value for
7.009232529961056e+24,2.5278726317240445e+24,2.6177112093242585e+23,1179713989619784000,7.790468414639561
e+24 (52ms)
      ✔ returns the correct value for
4.7176323676006554e+24,8.92150077078262e+24,2.6099000799690147e+24,1232714522999031600,2.883333545874649e
+24 (54ms)

  Comptroller
    _setActionsPaused
      ✔ reverts if the market is not listed
      ✔ does nothing if the actions list is empty
      ✔ does nothing if the markets list is empty
      ✔ can pause one action on several markets
      ✔ can pause several actions on one market (41ms)
      ✔ can pause and unpause several actions on several markets (104ms)

  assetListTest
    swapDebt
      ✔ fails if called by a non-owner
      ✔ fails if comptrollers don't match (57ms)
      ✔ fails if repayBorrowBehalf returns a non-zero error code (44ms)
      ✔ fails if borrowBehalf returns a non-zero error code (93ms)
      ✔ transfers repayAmount of underlying from the sender (125ms)
      ✔ approves vToken to transfer money from the contract (124ms)
      ✔ calls repayBorrowBehalf after transferring the underlying to self (112ms)
      ✔ converts the amounts using the oracle exchange rates (121ms)
      ✔ uses the actually repaid amount rather than specified amount (120ms)
      ✔ transfers the actually borrowed amount to the owner (122ms)
    sweepTokens
      ✔ fails if called by a non-owner
      ✔ transfers the full balance to the owner

  Evil Token test
Duplicate definition of Log (Log(string,address), Log(string,uint256))
Duplicate definition of Log (Log(string,address), Log(string,uint256))
Duplicate definition of Log (Log(string,address), Log(string,uint256))
    ✔ Check the updated vToken states after transfer out (737ms)

  Governor Bravo Cast Vote Test
    We must revert if:
      ✔ We cannot propose without enough voting power by depositing xvs to the vault
      after we deposit xvs to the vault
        ✔ There does not exist a proposal with matching proposal id where the current block number is
between the proposal's start block (exclusive) and end block (inclusive)
        ✔ Such proposal already has an entry in its voters set matching the sender (52ms)
        Otherwise
          ✔ we add the sender to the proposal's voters set (40ms)
        and we take the balance returned by GetPriorVotes for the given sender and the proposal's start
block, which may be zero,
          ✔ and we add that ForVotes (79ms)
          ✔ or AgainstVotes corresponding to the caller's support flag. (77ms)
        castVoteBySig
          ✔ reverts if the signatory is invalid
          ✔ casts vote on behalf of the signatory (84ms)

  Governor Bravo Initializing Test
    initilizer
      ✔ should revert if not called by admin
      ✔ should revert if invalid xvs address
      ✔ should revert if invalid guardian address
      ✔ should revert if timelock adress count differs from governance routes count
      ✔ should revert if proposal config count differs from governance routes count
      ✔ should revert if initialized twice (58ms)

  Governor Bravo Propose Tests
    simple initialization
      ✔ ID is set to a globally unique identifier
      ✔ Proposer is set to the sender
      ✔ Start block is set to the current block number plus vote delay
      ✔ End block is set to the current block number plus the sum of vote delay and vote period
      ✔ ForVotes and AgainstVotes are initialized to zero
      ✔ Executed and Canceled flags are initialized to false
      ✔ ETA is initialized to zero
      ✔ Targets, Values, Signatures, Calldatas are set according to parameters
      ✔ This function returns the id of the newly created proposal. # proposalId(n) = succ(proposalId(n-
1)) (41ms)
      ✔ emits log with id and description (78ms)
      This function must revert if
        ✔ the length of the values, signatures or calldatas arrays are not the same length, (82ms)
        ✔ or if that length is zero or greater than Max Operations.
        Additionally, if there exists a pending or active proposal from the same proposer, we must
revert.
          ✔ reverts with pending
          ✔ reverts with active

  Governor Bravo Queue Tests
    overlapping actions
      ✔ reverts on queueing overlapping actions in same proposal (90ms)
      ✔ reverts on queueing overlapping actions in different proposals (181ms)

  Governor Bravo State Tests
    ✔ Invalid for proposal not found
    ✔ Pending
    ✔ Active
    ✔ Canceled (78ms)
    ✔ Canceled by Guardian (67ms)
    ✔ Defeated
    ✔ Succeeded (92ms)
    ✔ Expired (134ms)
    ✔ Queued (132ms)
    ✔ Executed (197ms)

  XVS Vault Tests
    delegateBySig
      ✔ reverts if the market is paused
      ✔ reverts if the signatory is invalid
      ✔ reverts if the nonce is bad
      ✔ reverts if the signature has expired

  VenusLens: Rewards Summary
    ✔ Should get summary for all markets (240ms)

  Liquidator
    splitLiquidationIncentive
      ✔ splits liquidationIncentive between Treasury and Liquidator with correct amounts
    distributeLiquidationIncentive
      ✔ distributes the liquidationIncentive between Treasury and Liquidator with correct amounts (50ms)
      ✔ reverts if transfer to liquidator fails (41ms)
      ✔ reverts if transfer to treasury fails (49ms)

  Liquidator
    liquidateBorrow
      liquidating BEP-20 debt
        ✔ fails if borrower is zero address
        ✔ fails if some BNB is sent along with the transaction
        ✔ transfers tokens from the liquidator (173ms)
        ✔ approves the borrowed VToken to spend underlying (144ms)
        ✔ calls liquidateBorrow on borrowed VToken (145ms)
        ✔ transfers the seized collateral to liquidator and treasury (142ms)
        ✔ emits LiquidateBorrowedTokens event (140ms)
      liquidating VAI debt
        ✔ transfers VAI from the liquidator (154ms)
        ✔ approves VAIController to spend VAI (125ms)
        ✔ calls liquidateVAI on VAIController (123ms)
    liquidating BNB debt
      ✔ fails if msg.value is not equal to repayment amount (57ms)
      ✔ transfers BNB from the liquidator (79ms)
      ✔ calls liquidateBorrow on VBNB (77ms)
      ✔ forwards BNB to VBNB contract (78ms)
    setTreasuryPercent
      ✔ updates treasury percent in storage (50ms)
      ✔ fails when called from non-admin
      ✔ fails when the percentage is too high
      ✔ uses the new treasury percent during distributions (172ms)

  Liquidator
    Restricted liquidations
      addToAllowlist
        ✔ fails if called by a non-admin
        ✔ adds address to allowlist (44ms)
        ✔ fails if already in the allowlist (46ms)
        ✔ emits LiquidationPermissionGranted event
      removeFromAllowlist
        ✔ fails if called by a non-admin
        ✔ fails if not in the allowlist
        ✔ removes address from allowlist (79ms)
        ✔ emits LiquidationPermissionRevoked event (44ms)
      restrictLiquidation
        ✔ fails if called by a non-admin
        ✔ restricts liquidations for the borrower (41ms)
        ✔ fails if already restricted (64ms)
        ✔ emits LiquidationRestricted event
      unrestrictLiquidation
        ✔ fails if called by a non-admin
        ✔ removes the restrictions for the borrower (80ms)
        ✔ fails if not restricted
        ✔ emits LiquidationRestricted event (50ms)
      liquidateBorrow
        ✔ fails if the liquidation is restricted (48ms)
        ✔ proceeds with the liquidation if the guy is allowed to (89ms)

  Swap Contract
    ✔ revert if vToken address is not listed
    Setter
      ✔ should reverted if zero address
      ✔ should reverted if vToken not listed
      ✔ setting address for VBNBToken  (55ms)
    Swap
      ✔ revert if path length is 1
      ✔ revert if deadline has passed
      ✔ revert if output amoutn is below minimum
      ✔ should be reverted if tokenA == tokenB
      ✔ should swap tokenA -> tokenB (58ms)
      ✔ revert if deadline has passed
      ✔ revert if address zero
      ✔ should reverted if first address in not WBNB address
      ✔ should reverted if output amount is below minimum (51ms)
      ✔ should swap BNB -> token (70ms)
      ✔ revert if deadline has passed
      ✔ should swap tokenA -> tokenB  at supporting fee
      ✔ should reverted if deadline passed
      ✔ should swap BNB -> token  at supporting fee
      ✔ should swap EXact token -> BNB at supporting fee  (106ms)
      ✔ should swap tokesn for Exact BNB
      ✔ should swap tokens for Exact Tokens
      ✔ should swap tokens for Exact BNB
      ✔ should swap BNB for Exact Tokens
    Supply
      ✔ revert if deadline has passed
      ✔ swap tokenA -> tokenB --> supply tokenB (123ms)
      ✔ swap BNB -> token --> supply token (135ms)
      ✔ revert if deadline has passed at supporting fee
      ✔ swap tokenA -> tokenB --> supply tokenB at supporting fee (141ms)
      ✔ swap BNB -> token --> supply token at supporting fee (143ms)
      ✔ swap tokenA -> exact tokenB (120ms)
      ✔ swap bnb -> exact tokenB (166ms)
      ✔ Exact tokens -> BNB and supply
      ✔ Exact tokens -> BNB and supply at supporting fee
    Repay
      ✔ revert if deadline has passed
      ✔ swap tokenA -> tokenB --> supply tokenB (126ms)
      ✔ swap BNB -> token --> supply token (138ms)
      ✔ revert if deadline has passed at supporting fee
      ✔ swap tokenA -> tokenB --> reapy tokenB at supporting fee (143ms)
      ✔ swap BNB -> token --> repay token at supporting fee (140ms)
      ✔ swap tokenA -> exact tokenB (131ms)
      ✔ swap tokenA -> full debt of tokenB (133ms)
      ✔ swap bnb -> exact tokenB (144ms)
      ✔ swap bnb -> full tokenB debt (151ms)
      ✔ Exact tokens -> BNB at supporting fee (112ms)
      ✔ Exact tokens -> BNB (73ms)
      ✔ Tokens -> Exact BNB (73ms)
      ✔ Tokens -> Exact BNB and supply
      ✔ Tokens -> full debt of BNB
    Sweep Token
      ✔ Should be reverted if get zero address
      ✔ Sweep ERC-20 tokens (100ms)
    library function
      ✔ Quote function
      ✔ getAmoutIn function
      ✔ getAmoutout function
      ✔ getAmoutout function
      ✔ getAmoutout function

  admin / _setPendingAdmin / _acceptAdmin
    admin()
      ✔ should return correct admin
    pendingAdmin()
      ✔ should return correct pending admin
    _setPendingAdmin()
      ✔ should only be callable by admin (39ms)
      ✔ should properly set pending admin
      ✔ should properly set pending admin twice (50ms)
      ✔ should emit event
    _acceptAdmin()
      ✔ should fail when pending admin is zero (44ms)
      ✔ should fail when called by another account (e.g. root) (51ms)
      ✔ should succeed and set admin and clear pending admin (53ms)
      ✔ should emit log on success

  Unitroller
    constructor
      ✔ sets admin to caller and addresses to 0 (46ms)
    _setPendingImplementation
      Check caller is admin
        ✔ emits a failure log
        ✔ does not change pending implementation address
      succeeding
        ✔ stores pendingComptrollerImplementation with value newPendingImplementation
        ✔ emits NewPendingImplementation event
    _acceptImplementation
      Check caller is pendingComptrollerImplementation  and pendingComptrollerImplementation ≠ address(0)
        ✔ emits a failure log
        ✔ does not change current implementation address
      the brains must accept the responsibility of implementation
        ✔ Store comptrollerImplementation with value pendingComptrollerImplementation
        ✔ Unset pendingComptrollerImplementation
        ✔ Emit NewImplementation(oldImplementation, newImplementation)
        ✔ Emit NewPendingImplementation(oldPendingImplementation, 0)
      fallback delegates to brains
        ✔ forwards reverts
        ✔ gets addresses
        ✔ gets strings
        ✔ gets bools
      ✔ gets list of ints

  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 (55ms)
      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  (55ms)
          ✔ should revert if VAI transfer fails  (72ms)
          ✔ should revert if VAI to be burnt > vaiMinted  (62ms)
          should sucessfully perform the swap
            Fees: 10%
              ✔ stable token = 1$  (98ms)
              ✔ stable token < 1$  (101ms)
              ✔ stable token > 1$  (184ms)
            Fees: 0%
              ✔ stable token = 1$  (130ms)
              ✔ stable token < 1$  (87ms)
              ✔ stable token > 1$  (90ms)
        swapStableForVAI(address,uint256)
          ✔ should revert if receiver is zero address
          ✔ should revert if VAI mint cap will be reached  (86ms)
          ✔ should revert if amount after transfer is too small   (84ms)
          should sucessfully perform the swap
            Fees: 10%
              ✔ stable token = 1$  (123ms)
              ✔ stable token > 1$  (108ms)
              ✔ stable token < 1$  (111ms)
            Fees: 0%
              ✔ stable token = 1$  (103ms)
              ✔ stable token > 1$  (103ms)
              ✔ stable token < 1$  (112ms)
    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 (43ms)
          ✔ should revert if already paused
        resume()
          ✔ should revert if not authorised
          ✔ should resume if authorised
          ✔ should revert if already resumed (38ms)
        setFeeIn(uint256)
          ✔ should revert if not authorised
          ✔ should revert if fee is invalid
          ✔ set the correct fee (39ms)
        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 (75ms)
        setVenusTreasury(uint256)
          ✔ should revert if not authorised
          ✔ should revert if zero address (43ms)
          ✔ should set the treasury address (38ms)
        setOracle(address)
          ✔ should revert if not authorised
          ✔ should revert if oracle address is zero
          ✔ should set the oracle (57ms)
      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  (71ms)
          ✔ should revert if VAI transfer fails  (87ms)
          ✔ should revert if VAI to be burnt > vaiMinted  (74ms)
          should sucessfully perform the swap
            Fees: 10%
              ✔ stable token = 1$  (121ms)
              ✔ stable token < 1$  (125ms)
              ✔ stable token > 1$  (123ms)
            Fees: 0%
              ✔ stable token = 1$  (108ms)
              ✔ stable token < 1$  (106ms)
              ✔ stable token > 1$  (107ms)
        swapStableForVAI(address,uint256)
          ✔ should revert if receiver is zero address
          ✔ should revert if VAI mint cap will be reached  (104ms)
          should sucessfully perform the swap
            Fees: 10%
              ✔ stable token = 1$  (135ms)
              ✔ stable token > 1$  (146ms)
              ✔ stable token < 1$  (141ms)
            Fees: 0%
              ✔ stable token = 1$  (125ms)
              ✔ stable token > 1$  (124ms)
              ✔ stable token < 1$  (122ms)
    PSM: 6 decimals
      initialization
```

```
      ✔ should revert if contract already deployed
      ✔ should initialize sucessfully
      reverts if init address = 0x0:
        ✔ acm
        ✔ treasury
        ✔ stableToken (72ms)
      reverts if fee init value is invalid
        ✔ feeIn
        ✔ feeOut
    Admin functions
      pause()
        ✔ should revert if not authorised (43ms)
        ✔ should pause if authorised (41ms)
        ✔ should revert if already paused (44ms)
      resume()
        ✔ should revert if not authorised (55ms)
        ✔ should resume if authorised (42ms)
        ✔ should revert if already resumed (46ms)
      setFeeIn(uint256)
        ✔ should revert if not authorised (44ms)
        ✔ should revert if fee is invalid (40ms)
        ✔ set the correct fee (43ms)
      setFeeOut(uint256)
        ✔ should revert if not authorised (45ms)
        ✔ should revert if fee is invalid (45ms)
        ✔ set the correct fee (47ms)
      setVAIMintCap(uint256)
        ✔ should revert if not authorised (46ms)
        ✔ should set the correct mint cap (46ms)
      setVenusTreasury(uint256)
        ✔ should revert if not authorised (78ms)
        ✔ should revert if zero address (41ms)
        ✔ should set the treasury address (43ms)
      setOracle(address)
        ✔ should revert if not authorised (44ms)
        ✔ should revert if oracle address is zero (47ms)
        ✔ should set the oracle (70ms)
    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  (43ms)
        ✔ should revert if sender has insufficient VAI balance  (85ms)
        ✔ should revert if VAI transfer fails  (103ms)
        ✔ should revert if VAI to be burnt > vaiMinted  (85ms)
        should sucessfully perform the swap
          Fees: 10%
            ✔ stable token = 1$  (145ms)
            ✔ stable token < 1$  (152ms)
            ✔ stable token > 1$  (155ms)
          Fees: 0%
            ✔ stable token = 1$  (133ms)
            ✔ stable token < 1$  (183ms)
            ✔ stable token > 1$  (130ms)
      swapStableForVAI(address,uint256)
        ✔ should revert if receiver is zero address
        ✔ should revert if VAI mint cap will be reached  (128ms)
        should sucessfully perform the swap
          Fees: 10%
            ✔ stable token = 1$  (167ms)
            ✔ stable token > 1$  (169ms)
            ✔ stable token < 1$  (171ms)
          Fees: 0%
            ✔ stable token = 1$  (150ms)
            ✔ stable token > 1$  (155ms)
            ✔ stable token < 1$  (157ms)

VAIController
  ✔ check wallet usdt balance (42ms)
  #getMintableVAI
    ✔ oracle
```

```
      ✔ getAssetsIn
      ✔ getAccountSnapshot
      ✔ getUnderlyingPrice (88ms)
      ✔ getComtroller (45ms)
      ✔ success (241ms)
    #mintVAI
      ✔ success (403ms)
    #repayVAI
      ✔ success for zero rate (243ms)
      ✔ success for 1.2 rate repay all (357ms)
      ✔ success for 1.2 rate repay half (343ms)
    #getHypotheticalAccountLiquidity
      ✔ success for zero rate 0.9 vusdt collateralFactor (410ms)
      ✔ success for 1.2 rate 0.9 vusdt collateralFactor (520ms)
    #liquidateVAI
      ✔ liquidationIncentiveMantissa
      ✔ success for zero rate 0.2 vusdt collateralFactor (1263ms)
      ✔ success for 1.2 rate 0.3 vusdt collateralFactor (1378ms)
    #getVAIRepayRate
      ✔ success for zero baseRate (51ms)
      ✔ success for baseRate 0.1 floatRate 0.1 vaiPirce 1e18 (240ms)
      ✔ success for baseRate 0.1 floatRate 0.1 vaiPirce 0.5 * 1e18 (225ms)
    #getVAIRepayAmount
      ✔ success for zero rate (44ms)
      ✔ success for baseRate 0.1 floatRate 0.1 vaiPirce 1e18 (259ms)
      ✔ success for baseRate 0.1 floatRate 0.1 vaiPirce 0.5 * 1e18 (276ms)
    #getVAICalculateRepayAmount
      ✔ success for zero rate (66ms)
      ✔ success for baseRate 0.1 floatRate 0.1 vaiPirce 1e18 (411ms)
      ✔ success for baseRate 0.1 floatRate 0.1 vaiPirce 0.5 * 1e18 (430ms)
    #getMintableVAI
      ✔ include current interest when calculating mintable VAI (419ms)
    #accrueVAIInterest
      ✔ success for called once (155ms)
      ✔ success for called twice (225ms)
    #setBaseRate
      ✔ fails if access control does not allow the call (43ms)
      ✔ emits NewVAIBaseRate event (47ms)
      ✔ sets new base rate in storage (46ms)
    #setFloatRate
      ✔ fails if access control does not allow the call (46ms)
      ✔ emits NewVAIFloatRate event (47ms)
      ✔ sets new float rate in storage (45ms)
    #setMintCap
      ✔ fails if access control does not allow the call (44ms)
      ✔ emits NewVAIMintCap event (46ms)
      ✔ sets new mint cap in storage (44ms)
    #setReceiver
      ✔ fails if called by a non-admin
      ✔ reverts if the receiver is zero address
      ✔ emits NewVAIReceiver event
      ✔ sets VAI receiver address in storage
    #setAccessControl
      ✔ reverts if called by non-admin
      ✔ reverts if ACM is zero address
      ✔ emits NewAccessControl event
      ✔ sets ACM address in storage

  VAIVault
    ✔ claim reward (841ms)
    setVenusInfo
      ✔ fails if called by a non-admin
      ✔ fails if XVS address is zero
      ✔ fails if VAI address is zero
      ✔ disallows configuring tokens twice (50ms)

  VRTVault
    unit tests
      setLastAccruingBlock
        ✔ fails if ACM disallows the call (46ms)
        ✔ fails if trying to set lastAccuringBlock to some absurdly high value (44ms)
        ✔ fails if lastAccuringBlock has passed (90ms)
        ✔ fails if trying to set lastAccuringBlock to some past block (46ms)
        ✔ fails if trying to set lastAccuringBlock to the current block (46ms)
        ✔ correctly sets lastAccuringBlock to some future block (67ms)
        ✔ can move lastAccuringBlock to a later block (118ms)
        ✔ can move lastAccuringBlock to an earlier block (116ms)
        ✔ fails if trying to move lastAccuringBlock to a block in the past (91ms)
    scenario
      ✔ deposit (175ms)
      ✔ should claim reward (116ms)
      ✔ should not claim reward after certain block (148ms)

  XVSVault
    setXvsStore
      ✔ fails if XVS is a zero address
      ✔ fails if XVSStore is a zero address
      ✔ fails if the vault is already initialized
    add
      ✔ reverts if ACM does not allow the call (49ms)
      ✔ reverts if xvsStore is not set (48ms)
      ✔ reverts if a pool with this (staked token, reward token) combination already exists (68ms)
      ✔ reverts if staked token exists in another pool (48ms)
      ✔ reverts if reward token is a zero address (48ms)
      ✔ reverts if staked token is a zero address (46ms)
      ✔ reverts if alloc points parameter is zero (47ms)
      ✔ emits PoolAdded event (70ms)
      ✔ adds a second pool to an existing rewardToken (94ms)
      ✔ sets pool info (93ms)
      ✔ configures reward token in XVSStore (94ms)
    set
      ✔ reverts if ACM does not allow the call (45ms)
      ✔ reverts if pool is not found (46ms)
      ✔ reverts if total alloc points after the call is zero (69ms)
      ✔ succeeds if the pool alloc points is zero but total alloc points is nonzero (252ms)
      ✔ emits PoolUpdated event (72ms)
    setRewardAmountPerBlock
      ✔ reverts if ACM does not allow the call (47ms)
      ✔ reverts if the token is not configured in XVSStore (92ms)
      ✔ emits RewardAmountPerBlockUpdated event (93ms)
      ✔ updates reward amount per block (114ms)
    setWithdrawalLockingPeriod
      ✔ reverts if ACM does not allow the call (51ms)
      ✔ reverts if pool does not exist (46ms)
      ✔ reverts if the lock period is 0 (44ms)
      ✔ reverts if the lock period is absurdly high (45ms)
      ✔ emits WithdrawalLockingPeriodUpdated event (94ms)
      ✔ updates lock period (142ms)
    pendingReward
      ✔ includes the old withdrawal requests in the rewards computation (258ms)
      ✔ excludes the new withdrawal requests from the rewards computation (365ms)
    deposit
      ✔ reverts if the vault is paused (79ms)
      ✔ reverts if pool does not exist
      ✔ transfers pool token to the vault (128ms)
      ✔ updates user's balance (133ms)
      ✔ fails if there's a pre-upgrade withdrawal request (184ms)
      ✔ succeeds if the pre-upgrade withdrawal request has been executed (676ms)
      ✔ uses the safe _transferReward under the hood (336ms)
    executeWithdrawal
      ✔ fails if the vault is paused (69ms)
      ✔ only transfers the requested amount for post-upgrade requests (338ms)
      ✔ handles pre-upgrade withdrawal requests (354ms)
      ✔ handles pre-upgrade and post-upgrade withdrawal requests (553ms)
    requestWithdrawal
      ✔ fails if the vault is paused (77ms)
      ✔ transfers rewards to the user (299ms)
      ✔ uses the safe _transferReward under the hood (305ms)
      ✔ fails if there's a pre-upgrade withdrawal request (156ms)
    claim
      ✔ fails if there's a pre-upgrade withdrawal request (85ms)
      ✔ succeeds if the pre-upgrade withdrawal request has been executed (356ms)
      ✔ excludes pending withdrawals from the user's shares (426ms)
      ✔ correctly accounts for updates in reward per block (281ms)
      ✔ uses the safe _transferReward under the hood (188ms)
    _transferReward
      ✔ sends the available funds to the user (162ms)
      ✔ emits VaultDebtUpdated event if vault debt is updated (113ms)
      ✔ does not emit VaultDebtUpdated event if vault debt is not updated (129ms)
      ✔ records the pending transfer (128ms)
      ✔ records several pending transfers (263ms)
      ✔ sends out the pending transfers in addition to reward if full amount <= funds available (455ms)
      ✔ sends a part of the pending transfers and reward if full amount > funds available (429ms)
    pendingWithdrawalsBeforeUpgrade
      ✔ returns zero if there were no pending withdrawals
      ✔ returns zero if there is only a new-style pending withdrawal (167ms)
      ✔ returns the requested amount if there is an old-style pending withdrawal (48ms)
      ✔ returns the total requested amount if there are multiple old-style pending withdrawals (84ms)
      ✔ returns zero if the pending withdrawal was executed (194ms)
    Scenarios
      ✔ works correctly with multiple claim, deposit, and withdrawal requests (1331ms)


  525 passing (3m)

2. vBNBAdmin

  VBNBAdmin
The Hardhat Network tracing engine could not be initialized. Run Hardhat with --verbose to learn more.
    ✔ set VBNBAdmin as vBNB admin
    harvest income
      ✔ reduce BNB reserves (61ms)

  2 passing (3s)


3. Isolated pools

Total 174 tests passing

4. ProtocolShareReserve

  ProtocolShareReserve: Tests
    ✔ check configuration of schemas
    ✔ update configuration of schemas (52ms)
    ✔ collect and distribute of income (285ms)

  3 passing (2s)
```

Code Coverage

Coverage was only available for the `ProtocolShareReserve` contract, where branch coverage is 58.7%, which should be improved to reach at least 80% coverage.

1. Core pool

Coverage data is not available.

2. BNB income

Coverage data is not available.

3. Isolated pools

Coverage data is not available.

4. ProtocolShareReserve

File	% Lines	% Statements	% Branches	% Funcs
ProtocolShareReserve.sol	88.71% (**110**/124)	87.5% (**70**/80)	58.7% (**27**/46)	86.67% (**13**/15)

Changelog

- 2023-09-26 - Initial report
- 2023-10-12 - 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