Quantstamp

Venus Diamond Comptroller

Executive Summary

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

Type	Comptroller Diamond Proxy
Timeline	2023-08-07 through 2023-08-18
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Audit_scope_Diamond_Comptroller_August_6th,_2023_2bdb8893b85144
Source Code	• VenusProtocol/venus-protocol #728d618
Auditors	• Jennifer Wu Auditing Engineer • Julio Aguliar Auditing Engineer • Hytham Farah Auditing Engineer • Marius Guggenmos Senior Auditing Engineer

Documentation quality	High
Test quality	Medium
Total Findings	12 Acknowledged: 10 Mitigated: 2
High severity findings	0
Medium severity findings	0
Low severity findings	2 Acknowledged: 2
Undetermined severity findings	1 Acknowledged: 1
Informational findings	9 Acknowledged: 7 Mitigated: 2

Summary of Findings

Quantstamp has performed an audit on the `Comptroller` smart contracts, which have been refactored to the EIP-2535 Diamond Proxy pattern. The functionalities of the `Comptroller` have been divided into four facets: `PolicyFacet`, `SetterFacet`, `MarketFacet`, and `RewardFacet` with the original storage layout from `ComptrollerV11Storage`. In the EIP-2535 setup, the `Unitroller` delegates calls to the new `Diamond` contract. The `Diamond` contract identifies the correct facet address based on the function selector and delegates call correspondingly. The primary focus of the audit was to review the correctness of the EIP-2535 restructuring of the `Comptroller`.

The audit resulted in a total of 12 findings along with 9 best practices outlined below. We recommend addressing all identified issues.

Fix Review: Following the fix review, the client added a feature allowing privileged roles to approve and execute forced liquidations on markets. This feature has been updated in the issue VEN-8. We confirmed that all issues have either been fixed, mitigated, or acknowledged.

ID	DESCRIPTION	SEVERITY	STATUS
VEN-1	Incorrect Version of Solidity	Low	Acknowledged
VEN-2	Risk of Storage Collision in Array Handling	Low	Acknowledged
VEN-3	Facets Contain Redundant Functions	Informational	Acknowledged
VEN-4	Missing Input Validation	Informational	Mitigated
VEN-5	Unchecked Return Value of `isVToken()` in `_supportMarket` Function	Informational	Acknowledged
VEN-6	Delayed Invocation of `accrueInterest()` May Cause Reward Distribution Discrepancies	Informational	Acknowledged

ID	DESCRIPTION	SEVERITY	STATUS
VEN-7	Role-Based Access Control Offers Enhanced Security and Clarity	Informational	Acknowledged
VEN-8	Privileged Roles	Informational	Mitigated
VEN-9	Potential Out-of-Gas Error when Adding a Market	Informational	Acknowledged
VEN-10	Compound Empty Market Vulnerability	Informational	Acknowledged
VEN-11	Protocol Upgradability	Informational	Acknowledged
VEN-12	Fixed VAI Price May Expose Liquidation Process to Peg Instability	Undetermined	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.

Findings

VEN-1 Incorrect Version of Solidity

Low Acknowledged

Update

The client acknowledged the issue and provided the following explanation:

> Given that dynamic tuples are not involved in the encoding process of the core protocol, so no pressing need to update the version. Moreover, updating the version would introduce extensive changes to the code base. In this audit, our primary objective is to transition the architecture to Diamond without introducing new functionality or causing disruptions to the system.

Description: The contract currently uses solidity version 0.5.16. Due to a medium severity bug in the solidity compiler since version 0.5.8, this version is not recommended.

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

VEN-2 Risk of Storage Collision in Array Handling

Low | Acknowledged

Update

The client acknowledged the issue and provided the following explanation:

> If we're not aligning the version update with VEN-1, then there's no need for any changes related to this issue.

File(s) affected: `FacetBase.sol`

Description: In updated Solidity versions, specifically from ^0.6.0 onwards, developers are restricted from manually adjusting the length of storage arrays via operations like `array.length--`. This change was made to safeguard against potential storage collisions with large arrays.

We have noticed that the `MarketFacet.exitMarket()` function currently utilizes such a pattern:

```
for (; i < len; ++i) {
    if (userAssetList[i] == vToken) {
        userAssetList[i] = userAssetList[len - 1];
        userAssetList.length--;
        break;
    }
}
```

Recommendation: To ensure safety and forward compatibility with newer Solidity versions, consider refactoring the way elements are removed from arrays. Specifically, use the `.pop()` method for this purpose. Adopting this recommendation also ensures that the code will not encounter compiler errors if the Solidity version is upgraded.

VEN-3 Facets Contain Redundant Functions

Informational | Acknowledged

Update

The client acknowledged the issue and provided the following explanation:

> There's no requirement to convert external functions to internal ones, considering there are only four of them. Among these, three are either view or pure functions.

File(s) affected: `ComptrollerStorage.sol`, `FacetBase.sol`

Description: Any functions that are declared as public or external in the `FacetBase` contract will be included in the bytecode of each facet since each facet inherits from this contract. Similarly, every state variable declared as public in the `ComptrollerStorage.sol` file will generate a getter function in every facet. This will unnecessarily bloat the bytecode of the implementation contracts, making deployments more expensive and slightly increasing runtime gas costs from the increase in available functions.

Recommendation: Declare all of the functions in the `FacetBase` contract as internal. In case any of these functions should be exposed, add a public or external wrapper that calls the internal function in one of the facets. Additionally, change the visibility of all variables in the `ComptrollerStorage.sol` file to internal. In case view functions are required, add them manually to one of the facets.

VEN-4 Missing Input Validation

Informational | Mitigated

Update

The client addressed 2 of the missing validations in `c60497ab220cb483b75df242ffd4fe08439a438e`.

The client provided the following explanation:

1. SetterFacet._setCloseFactor(). ⇒ The changes are done
2. SetterFacet._setLiquidationIncentive(). ⇒ This can be reviewed in the VIP
3. SetterFacet._setLiquidatorContract(). ⇒ The changes are done
4. SetterFacet._setTreasuryData(). Skip it. ⇒This can be reviewed in the VIP
5. SetterFacet._setVenusVAIVaultRate(). ⇒ This can be reviewed in the VIP.
6. SetterFacet._setVAIVaultInfo(). ⇒ This can be reviewed in the VIP.

File(s) affected: `SetterFacet.sol`

Description: It is crucial to validate inputs, even if the inputs come from trusted addresses, to avoid human error. A lack of robust input validation can only increase the likelihood and impact in the event of mistakes.

Following is the list of places that can potentially benefit from stricter input validation:

1. Fixed `SetterFacet._setCloseFactor()` : Update the function to only accept `newCloseFactorMantissa` values between `closeFactorMinMantissa` and `closeFactorMaxMantissa` .
2. Acknowledged `SetterFace._setLiquidationIncentive()` : Introduce an upper bound for `newLiquidationIncentiveMantissa` .
3. Fixed `SetterFacet._setLiquidatorContract()` : Implement an `ensureNonzeroAddress()` check for `newLiquidatorContract_` to ensure it is not set to a zero address.
4. Acknowledged `SetterFacet._setTreasuryData()` : Besides the check for values < 1e18, consider including a definitive cap for `newTreasurePercent` .
5. Acknowledged `SetterFacet._setVenusVAIVaultRate()` : Define and apply upper and lower limits for the new rate.
6. Acknowledged `SetterFacet._setVAIVaultInfo()` : Set restrictions on `releaseStartBlock_` and define a minimum acceptable value for `minReleaseAmount_` .

Recommendation: Add the validations and checks listed in the description.

VEN-5
Unchecked Return Value of `isVToken()` in `_supportMarket` Function

Informational | Acknowledged

Update

The client acknowledged the issue and provided the following explanation:

> Unfortunately, making changes isn't feasible. If a contract doesn't have an isVToken method, altering the code to check for a false value would lead to code breakage. Additionally, isVToken is a constant boolean, so modifying it wouldn't be a suitable solution either.

File(s) affected: `MarketFacet.sol`

Description: The function `_supportMarket(VToken vToken)` calls the `isVToken()` function, which is intended to serve as a sanity check to verify that the provided token is indeed a `VToken` . The `isVToken()` function returns a boolean value indicating the result of this check. However, the return value of `isVToken()` is not being checked or acted upon in the `_supportMarket` function. This oversight means that even if the provided token is not a `VToken` , the `_supportMarket` function will continue its execution as long as the token is listed.

Recommendation: Ensure that the return value of `isVToken()` is properly checked within the `_supportMarket` function. If `isVToken()` returns `false` , the function should immediately revert with an appropriate error message indicating that the provided token is not a valid `VToken` .

VEN-6
Delayed Invocation of `accrueInterest()` May Cause Reward Distribution Discrepancies

Informational | Acknowledged

Update

The client acknowledged the issue and provided the following explanation:

> Given that accrueInterest() will be invoked before any interaction with the vToken, it's likely that the block delta won't be a significantly large number. Considering this we acknowledge the situation for now.

File(s) affected: `XVSRewardsHelper.sol` , `PolicyFacet.sol` , `RewardFacet.sol`

Description: The `XVSRewardsHelper.updateVenusBorrowIndex()` function references the state variables `VToken.borrowIndex` and `Vtoken.totalBorrows`, which are updated in the `VToken.accrueInterest()` function. If `accrueInterest()` is not invoked in a timely manner, especially after significant block intervals, these state variables could be outdated during the reward distribution processes.

In the process of computing the `borrowAmount` in `updateVenusBorrowIndex()`, an outdated `borrowIndex` can lead to a stale `borrowAmount`. Consequently, this may affect the reward distribution, causing users to receive rewards that deviate from the intended amounts.

The `updateVenusBorrowIndex()` is used in the following functions:

1. `PolicyFacet.repayBorrowAllowed()`
2. `PolicyFacet.borrowAllowed()`
3. `RewardFacet.claimVenus()`

Recommendation: This issue is written as an informational note that while the `borrowIndex` typically updates alongside changes in `borrowAmount`, the variables may become stale due to the passage of significant blocks without the invocation of `accrueInterest()`. To mitigate potential discrepancies in reward distributions, it is suggested to either ensure that `venusBorrowState[vToken].block` is updated in alignment with `VToken.accrualBlockNumber` or invoke `accrueInterest()` prior to the calculation of venus rewards.

VEN-7
Role-Based Access Control Offers Enhanced Security and Clarity

Informational | Acknowledged

Update

The client acknowledged the issue and provided the following explanation:

> This would entail a significant alteration since it would necessitate a complete overhaul of our access system, requiring the granting of all permissions through VIP. Currently, we are not looking to make such big changes in this Audit scope.

File(s) affected: `FacetBase.sol`

Description: The current implementation uses the `FacetBase.ensureAllowed()` method to validate access based on a function's string signature. This approach can lead to potential errors due to misspellings or changes in function signatures and does not provide a clear hierarchy of permissions.

This current approach can introduce vulnerabilities and confusion in multiple scenarios:

- Misspellings or Syntax Errors: A typo or a slight variation in the function signature string can lead to unintended privilege grants.
- Stale Permissions: As facets or functions update, the string-based permissions can become stale.
- Unintended Privilege Grants: If an existing function is removed and a new function is added with an identical function signature, using its string signature without updating the permissions can unintentionally grant or restrict access.

Recommendation: For increased clarity and security, consider implementing a role-based access control mechanism. Here are some steps to move in that direction:

1. Define Roles: Clearly define roles like `ADMIN`, `MAINTAINER`, `UPGRADER`, etc., which describe the responsibilities and permissions associated with them.
2. Assign Roles: Assign these roles to specific addresses using an `AccessControlManager`. This way, permissions are managed based on roles rather than individual function signatures.
3. Update Permission Checks: In each function that requires permission checks, validate the caller's role instead of their ability to call a specific function.
4. Maintain Flexibility: Design the access control system such that roles can be easily granted, revoked, and queried. This ensures that as the protocol evolves, permissions can be adjusted without major code changes.

Using role-based access control simplifies the permission model and provides a clearer understanding of who can do what, making the system more transparent and secure.

VEN-8 Privileged Roles

Informational | Mitigated

Update

The client updated the code documentation to reflect the correct privileged permissions of admin or privileged users in `b745623f6bb95b978b8bd3c62fac6fcff5ac277d` and `456ad851313494ffa716c49a8a0cc851c09b9130`.

File(s) affected: `Unitroller.sol`, `Diamond.sol`, `PolicyFacet.sol`, `RewardFacet.sol`, `SetterFacet.sol`

Description: All privileged permissions should be clearly documented for users. The privileged permissions of the `Comptroller` contract are documented below.

The following are cases of contract `admin` privileges:

1. The `Unitroller` is an upgradeable contract that can have its implementation contracts swapped out at any time through the `_setPendingImplementation()` function by the contract `admin`.
2. The `Diamond` contract can add, remove, and update any function selectors and facet addresses through `diamondCut()` function by the contract `admin`.
3. The function `PolicyFacet._setVenusSpeeds()` allows the contract `admin` to set XVS speed for a market.
4. The function `RewardFacet._grantXVS()` allows the contract `admin` to transfer XVS to any recipient based on the recipient's shortfall.
5. The function `SetterFacet._setPriceOracle()` allows the contract `admin` to set a new price oracle used by the `Comptroller`.
6. The function `SetterFacet._setCloseFactor()` allows the contract `admin` to set the `closeFactor` used to liquidate borrows.
7. The function `SetterFacet._setAccessControl()` allows the contract `admin` to set the address of access control used to verify role privileges.
8. The function `SetterFacet._setLiquidatorContract()` allows the contract `admin` to set the liquidation contract.
9. The function `SetterFacet._setPauseGuardian()` allows the contract `admin` to set the pause guardian contract.

The following are cases of role privileges:

1. The function `MarketFacet._supportMarket()` allows a privileged role to add and list markets to the `Comptroller`.
2. The function `SetterFacet._setCollateralFactor()` allows a privileged role to set the `collateralFactorMantissa`.
3. The function `SetterFacet._setLiquidationIncentive` allows a privileged role to set the `liquidationIncentiveMantissa`.
4. The function `SetterFacet._setMarketBorrowCaps()` allows a privileged role to set the borrowing cap for a vToken market.
5. The function `SetterFacet._setMarketSupplyCaps()` allows a privileged role to set the supply cap for a vToken.
6. The function `SetterFacet._setProtocolPaused()` allows a privileged role to pause/unpause protocol.
7. The function `SetterFacet._setActionsPaused()` allows a privileged role to pause/unpause the protocol action state.
8. (**Fix Review**) The function `SetterFacet._setForcedLiquidation()` allows a privileged role to enable/disable force liquidation for a vToken market. If forced liquidation is enabled, borrows in the market may be liquidated regardless of the account liquidity.

Recommendation: Clarify the impact of these privileged actions on the end-users via publicly facing documentation.

VEN-9 Potential Out-of-Gas Error when Adding a Market

Informational | Acknowledged

Update

The client acknowledged the issue and provided the following explanation:

> It will go out of gas only when there are a large number of markets but currently, we only have around 30 markets in the core pool. So, this error will not occur. The Audit's scope is to transition the architecture to Diamond with minimal changes to the code base. We acknowledge these changes and can address the limits in upcoming releases.

File(s) affected: `MarketFacet.sol`

Description: The `_addMarketInternal()` function uses a loop to iterate over all existing markets to verify that the market is not already listed. If the array of markets grows significantly, this loop could consume excessive gas, leading to an out-of-gas error during execution.

Recommendation:

1. Conduct a gas analysis for `_addMarketInternal()` to understand its behavior with increasing market count. Based on the analysis, consider imposing a limit on the total number of markets that can be added.
2. Alternatively, for better optimization, reconsider the linear search approach for verifying an existing vToken market. Utilizing a mapping for direct lookup can be more gas efficient and eliminate the need for iteration.

VEN-10 Compound Empty Market Vulnerability

Informational | Acknowledged

Update

The client acknowledged the issue and provided the following explanation:

> As we usually do mintBehalf immediately after we call _supportMarket. To provide initial liquidity to the new markets and make sure the similar scenario is not hit.

Description: Venus Protocol, a fork of Compound, may be susceptible to a vulnerability identified in Compound V2 when a market is empty. The empty market vulnerability is a result of two interlinked bugs:

1. The exchange rate of cTokens (or vTokens in the case of Venus) can be manipulated as it is calculated based on the contract's underlying asset balance in the function `getCashPrior()`.
2. A rounding error during the redemption process allows attackers to redeem their underlying assets after borrowing funds without their account becoming insolvent in `redeemUnderlying()`.

Recommendation:

1. **Mint Small vToken Amounts during Market Creation:** Whenever launching a new market, mint a nominal number of vTokens (cTokens in Compound) and subsequently burn them. This ensures that the total supply never reaches zero, a state that makes the protocol vulnerable.

Specifically, when listing a new collateral token, initially set its collateral factor to zero in the Comptroller, mint some vTokens, burn them, and only then adjust the collateral factor to the intended value.

2. **Regular Monitoring and Maintenance of Existing Markets:** Given that existing markets can gradually deplete over time, periodic checks should be made. If a market is found near empty, the aforementioned process of minting and burning vTokens should be executed to prevent the total supply from dropping to zero.

VEN-11 Protocol Upgradability

Informational | Acknowledged

> **Update**
> The client acknowledged the issue and provided the following explanation:
>
> > The upgrades are done via VIP, which the community has to vote for in order to execute them.

Description: While upgradability is not a vulnerability in itself, users should be aware that the `Comptroller` implementation and facet contracts can be upgraded at any given time by the contract `admin`. This audit does not guarantee the behavior of future upgraded contracts.

Recommendation: The fact that the contracts can be upgraded and reasons for future upgrades should be communicated to users prior to upgrading.

VEN-12 Fixed VAI Price May Expose Liquidation Process to Peg Instability

Undetermined | Acknowledged

> **Update**
> The client acknowledged the issue and provided the following explanation:
>
> > We consider this potential change out of the scope of the main changes (upgrade the “distribution” of the code in the Comptroller contract), and we prefer to keep it as it is now. We’ll analyze the use of an oracle for the VAI price during liquidations (and minting) in the future

File(s) affected: `ComptrollerLens.sol`

Description: The `ComptrollerLens.liquidateVAICalculateSeizeTokens()` function determines the VAI liquidation process. The VAI price is statically hard-coded to 1e18. Although the protocol has plans to incorporate a feature that permits pausing VAI liquidations as a safeguard against potential de-pegging, this mitigation strategy does not have an automated trigger. Without such automation, there's an inherent risk of unintended repercussions for the protocol and its users in the event of VAI de-pegging.

Recommendation: Instead of using a fixed VAI price, consider integrating an Oracle mechanism to retrieve the current and real-time price of VAI. This ensures that the liquidation process accurately reflects the market status and mitigates risks associated with peg instability.

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

The code documentation for the functions should be updated to reflect its access. Set the following functions to restricted access because they are using `ensureAllowed()`:

1. Fixed `SetterFacet._setLiquidationIncentive()`
2. Fixed `SetterFacet._setMarketBorrowCaps()`
3. Fixed `SetterFacet._setMarketSupplyCaps()`
4. Fixed `SetterFacet._setActionsPaused()`

Adherence to Best Practices

1. Acknowledged It is recommended to use explicit type widths (i.e. `uint256`) over implicit ones; consider updating `uint` to `uint256` in `ComptrollerInterface.sol`.
2. Acknowledged Consider having the `ComptrollerInterface` directly inherit from all the facets' interfaces.
3. Acknowledged When casting from `uint256` to smaller integer types, there's a risk of value truncation if the larger value exceeds the capacity of the smaller type. To mitigate this, consider using OpenZeppelin's `SafeCast` library for safe type conversions. The following instances casts `uint256` to small integer types without validation:
    1. `Diamond.sol#L117`
    2. `Diamond.sol#L140`
    3. `MarketFacet.sol#L242`
    4. `XVSRewardHelper.sol#L44`
    5. `XVSRewardHelper.sol#L46`
    6. `XVSRewardHelper.sol#L68`
    7. `XVSRewardHelper.sol#L70`
4. Acknowledged The function parameter `facetAddress` shadows the function `Diamond.facetAddress()`; consider renaming the variable to avoid shadow conflicts in the following functions:
    1. `Diamond.addFunctions()`
    2. `Diamond.replaceFunctions()`
    3. `Diamond.removeFunctions()`
    4. `Diamond.addFacet()`
    5. `Diamond.addFunction()`
    6. `Diamond.removeFunction()`
5. Acknowledged Consider validating the current total supply of the vToken before setting the new supply cap in the function `StterFacet._setMarketSuppyCaps()`.
6. Acknowledged Update the `VAIControllerInterface` since is outdated and contains several functions that are not present in the `VAIController` contract. Consider inheriting from the interface in the implementation contract to ensure all functions are present.
7. Acknowledged Public functions that are not called by the contract itself should be declared as `external`, and their immutable parameters should be placed in `calldata` to optimize gas usage. The `Diamond.diamondCut()` function should have `external` visibility, and its `diamondCut_` parameter's location should be changed to `calldata`.
8. Acknowledged Consider using more custom errors and fewer require statements (once the protocol has been upgraded to a more recent version of solidity). They are more gas efficient as well as allowing for more intricate error handling logic.
9. Acknowledged In the `MarketFacet.exitMarket()` function it is recommended to check if the user is already in the market earlier in the function.
10. Acknowledged Consider converting the following validating input functions to modifiers:
    1. `ensureListed()`
    2. `ensureAdmin()`
    3. `ensureAdminOr()`
    4. `ensureNonzeroAddress()`
    5. `checkActionPauseState()`

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Contracts

- `ff9...409 ./Lens/SnapshotLens.sol`
- `188...8cd ./Lens/ComptrollerLens.sol`
- `c4b...367 ./Comptroller/ComptrollerInterface.sol`
- `197...e1c ./Comptroller/Unitroller.sol`
- `aa4...8dd ./Comptroller/ComptrollerLensInterface.sol`
- `3c9...d51 ./Comptroller/ComptrollerStorage.sol`
- `e8f...258 ./Comptroller/Diamond/Diamond.sol`
- `a2e...cee ./Comptroller/Diamond/interfaces/IMarketFacet.sol`

- `2ad...01e ./Comptroller/Diamond/interfaces/IDiamondCut.sol`
- `f3a...4ac ./Comptroller/Diamond/interfaces/ISetterFacet.sol`
- `e5e...edc ./Comptroller/Diamond/interfaces/IRewardFacet.sol`
- `b09...8ae ./Comptroller/Diamond/interfaces/IPolicyFacet.sol`
- `197...b21 ./Comptroller/Diamond/facets/MarketFacet.sol`
- `5b7...c92 ./Comptroller/Diamond/facets/PolicyFacet.sol`
- `c89...40e ./Comptroller/Diamond/facets/FacetBase.sol`
- `f78...026 ./Comptroller/Diamond/facets/XVSRewardsHelper.sol`
- `f48...70d ./Comptroller/Diamond/facets/RewardFacet.sol`
- `3b6...00a ./Comptroller/Diamond/facets/SetterFacet.sol`

Tests

- `79f...098 ./Comptroller/protocolPauseTest.js`
- `785...c57 ./Comptroller/accountLiquidityTest.js`
- `883...2fa ./Comptroller/releaseToVaultTest.js`
- `195...dd1 ./Comptroller/proxiedComptrollerTest.js`

Toolset

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

Setup

Tool Setup:

- Slither v0.9.6

Steps taken to run the tools:

1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither . --filter-paths "DelegateBorrowers|Governance|InterestRateModels|Lens/VenusLens.sol|Lens/XVSStakingLens.sol|Liquidator|Oracle|Swap|test|Tokens|Utils|Vault|VRTVault|XVSVault|PegStability"`

Automated Analysis

Slither

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

Test Suite Results

The code coverage was generated for contracts in contracts/Comptroller using `npx hardhat coverage` and .solcover.js provided below.

The solidity code coverage was limited with .solcover.js file set to:

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

```
  Comptroller
    _initializeMarket
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 (40ms)
      ✔ Updating non-zero speeds after setting it zero (90ms)
```

```
Comptroller
  _setAccessControlManager
    ✔ Reverts if called by non-admin
    ✔ Reverts if ACM is zero address
    ✔ Sets ACM address in storage
    ✔ should revert on same value (38ms)
  Access Control
    setCollateralFactor
      ✔ Should have AccessControl
      ✔ Should revert for same values
    setLiquidationIncentive
      ✔ Should have AccessControl
    setMarketBorrowCaps
      ✔ Should have AccessControl
    setMarketSupplyCaps
      ✔ Should have AccessControl
    setProtocolPaused
      ✔ Should have AccessControl (56ms)
    setActionsPaused
      ✔ Should have AccessControl
    supportMarket
      ✔ Should have AccessControl

Comptroller: assetListTest
  enterMarkets
    ✔ properly emits events (98ms)
    ✔ adds to the asset list only once (397ms)
    ✔ the market must be listed for add to succeed (100ms)
    ✔ returns a list of codes mapping to user's ultimate membership in given addresses (77ms)
  exitMarket
    ✔ doesn't let you exit if you have a borrow balance (104ms)
    ✔ rejects unless redeem allowed (161ms)
    ✔ accepts when you're not in the market already (105ms)
    ✔ properly removes when there's only one asset (139ms)
    ✔ properly removes when there's only two assets, removing the first (159ms)
    ✔ properly removes when there's only two assets, removing the second (165ms)
    ✔ properly removes when there's only three assets, removing the first (181ms)
    ✔ properly removes when there's only three assets, removing the second (194ms)
    ✔ properly removes when there's only three assets, removing the third (177ms)
  entering from borrowAllowed
    ✔ enters when called by a vtoken (50ms)
    ✔ reverts when called by not a vtoken (45ms)
    ✔ adds to the asset list only once (117ms)

Comptroller
  constructor
    ✔ on success it sets admin to creator and pendingAdmin is unset (915ms)
  _setLiquidationIncentive
    ✔ fails if incentive is less than 1e18
    ✔ accepts a valid incentive and emits a NewLiquidationIncentive event
    ✔ should revert on same values (41ms)
  _setVenusVAIVaultRate
    ✔ should revert on same values
  _setVAIVaultInfo
    ✔ should revert on same values (41ms)
  _setVAIController
    ✔ should revert on same values (38ms)
  _setVAIMintRate
    ✔ should revert on same values
  _setLiquidatorContract
    ✔ should revert on same values
    ✔ should revert on zero address (39ms)
  _setPauseGuardian
    ✔ should revert on same values (39ms)
  _setVenusSpeeds
    ✔ ensure non zero address for venus speeds
  _setPriceOracle
    ✔ fails if called by non-admin
    ✔ accepts a valid price oracle and emits a NewPriceOracle event
    ✔ Should revert on same values
  _setComptrollerLens
```

```
    ✔ fails if not called by admin
    ✔ should fire an event
    ✔ should revert on same value (40ms)
  _setCloseFactor
    ✔ fails if not called by admin
    ✔ should revert on same values
    ✔ fails if factor is set out of range
  _setCollateralFactor
    ✔ fails if asset is not listed
    ✔ fails if factor is set without an underlying price (46ms)
    ✔ succeeds and sets market (46ms)
    ✔ should revert on same values (64ms)
  _setForcedLiquidation
    ✔ fails if asset is not listed (59ms)
    ✔ fails if ACM does not allow the call
    ✔ sets forced liquidation (81ms)
    ✔ sets forced liquidation for VAI, even though it is not a listed market (81ms)
    ✔ emits IsForcedLiquidationEnabledUpdated event (46ms)
  _supportMarket
    ✔ fails if asset is not a VToken
    ✔ succeeds and sets market
    ✔ cannot list a market a second time (47ms)
    ✔ can list two different markets (51ms)
  Hooks
  mintAllowed
    ✔ allows minting if cap is not reached
    ✔ reverts if supply cap reached (78ms)
    ✔ reverts if market is not listed
  redeemVerify
    ✔ should allow you to redeem 0 underlying for 0 tokens (366ms)
    ✔ should allow you to redeem 5 underlyig for 5 tokens (369ms)
    ✔ should not allow you to redeem 5 underlying for 0 tokens (367ms)
  liquidateBorrowAllowed
    isForcedLiquidationEnabled == true
      ✔ reverts if borrowed market is not listed
      ✔ reverts if collateral market is not listed
      ✔ does not revert if borrowed vToken is VAIController (48ms)
      ✔ allows liquidations without shortfall
      ✔ allows to repay 100% of the borrow
      ✔ fails with TOO_MUCH_REPAY if trying to repay > borrowed amount
      ✔ checks the shortfall if isForcedLiquidationEnabled is set back to false
    isForcedLiquidationEnabled == false
      ✔ reverts if borrowed market is not listed
      ✔ reverts if collateral market is not listed
      ✔ does not revert if borrowed vToken is VAIController
      ✔ fails if borrower has 0 shortfall
      ✔ succeeds if borrower has nonzero shortfall

    Comptroller
      ✔ Revert on check for the function selector (60ms)
      ✔ Add Facet and function selectors to proxy (75ms)
      ✔ Get all facet function selectors by facet address
      ✔ Get facet position by facet address
      ✔ Get all facet addresses
      ✔ Get all facets address and their selectors
      ✔ Get facet address and position by function selector
      ✔ Remove function selector from facet mapping (55ms)
      ✔ Replace the function from facet mapping (96ms)
      ✔ Remove all functions (82ms)

    Comptroller
      liquidateCalculateAmountSeize
        ✔ fails if borrowed asset price is 0
        ✔ fails if collateral asset price is 0
        ✔ fails if the repayAmount causes overflow
        ✔ fails if the borrowed asset price causes overflow
        ✔ reverts if it fails to calculate the exchange rate
        ✔ returns the correct value for
100000000000000000,100000000000000000,100000000000000000,100000000000000000,100000000000000000
(46ms)
        ✔ returns the correct value for
200000000000000000,100000000000000000,100000000000000000,100000000000000000,100000000000000000
```

(44ms)
      ✔ returns the correct value for
200000000000000000,200000000000000000,142000000000000000,130000000000000000,245000000000000000
(42ms)
      ✔ returns the correct value for
278900000000000000,523048084200000000,771320000000000000000,130000000000000000,1.000245e+22 (44ms)
      ✔ returns the correct value for
7.009232529961056e+24,2.5278726317240445e+24,2.6177112093242585e+23,1179713989619784000,7.790468414639561
e+24 (43ms)
      ✔ returns the correct value for
9.157587319676866e+24,6.849897529358631e+24,6.742356911686356e+24,1336570065369529300,5.325327555698168e+
24 (46ms)

  ComptrollerMock
    _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 (40ms)
      ✔ can pause several actions on one market (47ms)
      ✔ can pause and unpause several actions on several markets (111ms)

  assetListTest
    swapDebt
      ✔ fails if called by a non-owner
      ✔ fails if comptrollers don't match (47ms)
      ✔ fails if repayBorrowBehalf returns a non-zero error code (42ms)
      ✔ fails if borrowBehalf returns a non-zero error code (69ms)
      ✔ transfers repayAmount of underlying from the sender (66ms)
      ✔ approves vToken to transfer money from the contract (78ms)
      ✔ calls repayBorrowBehalf after transferring the underlying to self (77ms)
      ✔ converts the amounts using the oracle exchange rates (74ms)
      ✔ uses the actually repaid amount rather than specified amount (94ms)
      ✔ transfers the actually borrowed amount to the owner (89ms)
    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 (434ms)

  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 (78ms)
      Otherwise
        ✔ we add the sender to the proposal's voters set (39ms)
      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 (134ms)
        ✔ or AgainstVotes corresponding to the caller's support flag. (88ms)
      castVoteBySig
        ✔ reverts if the signatory is invalid
        ✔ casts vote on behalf of the signatory (130ms)

  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 (69ms)

  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)) (46ms)
      ✔ emits log with id and description (87ms)
      This function must revert if
        ✔ the length of the values, signatures or calldatas arrays are not the same length, (106ms)
        ✔ 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 (107ms)
      ✔ reverts on queueing overlapping actions in different proposals (209ms)

Governor Bravo State Tests
  ✔ Invalid for proposal not found
  ✔ Pending
  ✔ Active
  ✔ Canceled (81ms)
  ✔ Canceled by Guardian (104ms)
  ✔ Defeated
  ✔ Succeeded (108ms)
  ✔ Expired (145ms)
  ✔ Queued (147ms)
  ✔ Executed (199ms)

XVS Vault Tests
  delegateBySig
    ✔ reverts if the market is paused (48ms)
    ✔ 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 (146ms)

Liquidator
  splitLiquidationIncentive
      ✔ splits liquidationIncentive between Treasury and Liquidator with correct amounts
    distributeLiquidationIncentive
      ✔ distributes the liquidationIncentive between Treasury and Liquidator with correct amounts (41ms)
      ✔ reverts if transfer to liquidator fails
      ✔ reverts if transfer to treasury fails (39ms)

  Liquidator
    liquidateBorrow
      liquidating BEP-20 debt
network block skew detected; skipping block events (emitted=2472 blockNumber3509)
        ✔ fails if borrower is zero address
        ✔ fails if some BNB is sent along with the transaction
        ✔ transfers tokens from the liquidator (112ms)
network block skew detected; skipping block events (emitted=2475 blockNumber3512)
        ✔ approves the borrowed VToken to spend underlying (92ms)
        ✔ calls liquidateBorrow on borrowed VToken (94ms)
        ✔ transfers the seized collateral to liquidator and treasury (92ms)
        ✔ emits LiquidateBorrowedTokens event (95ms)
      liquidating VAI debt
        ✔ transfers VAI from the liquidator (105ms)
        ✔ approves VAIController to spend VAI (90ms)
        ✔ calls liquidateVAI on VAIController (85ms)
    liquidating BNB debt
      ✔ fails if msg.value is not equal to repayment amount (62ms)
```

```
    ✔ transfers BNB from the liquidator (61ms)
    ✔ calls liquidateBorrow on VBNB (60ms)
    ✔ forwards BNB to VBNB contract (62ms)
  setTreasuryPercent
    ✔ updates treasury percent in storage (51ms)
    ✔ fails when called from non-admin
    ✔ fails when the percentage is too high
    ✔ uses the new treasury percent during distributions (132ms)

Liquidator
  Restricted liquidations
    addToAllowlist
      ✔ fails if called by a non-admin
      ✔ adds address to allowlist (41ms)
      ✔ fails if already in the allowlist (54ms)
      ✔ emits LiquidationPermissionGranted event
    removeFromAllowlist
      ✔ fails if called by a non-admin
      ✔ fails if not in the allowlist
      ✔ removes address from allowlist (77ms)
      ✔ emits LiquidationPermissionRevoked event (56ms)
    restrictLiquidation
      ✔ fails if called by a non-admin
      ✔ restricts liquidations for the borrower (41ms)
      ✔ fails if already restricted (65ms)
      ✔ emits LiquidationRestricted event
    unrestrictLiquidation
      ✔ fails if called by a non-admin
      ✔ removes the restrictions for the borrower (77ms)
      ✔ fails if not restricted
      ✔ emits LiquidationRestricted event (54ms)
    liquidateBorrow
      ✔ fails if the liquidation is restricted (57ms)
      ✔ proceeds with the liquidation if the guy is allowed to (96ms)

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  (60ms)
  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 (59ms)
    ✔ 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 (48ms)
    ✔ should swap BNB -> token (63ms)
    ✔ 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  (103ms)
    ✔ 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 (96ms)
    ✔ swap BNB -> token --> supply token (109ms)
    ✔ revert if deadline has passed at supporting fee
    ✔ swap tokenA -> tokenB --> supply tokenB at supporting fee (110ms)
    ✔ swap BNB -> token --> supply token at supporting fee (96ms)
    ✔ swap tokenA -> exact tokenB (93ms)
    ✔ swap bnb -> exact tokenB (172ms)
    ✔ Exact tokens -> BNB and supply
    ✔ Exact tokens -> BNB and supply at supporting fee
```

```
Repay
  ✔ revert if deadline has passed
  ✔ swap tokenA -> tokenB --> supply tokenB (99ms)
  ✔ swap BNB -> token --> supply token (101ms)
  ✔ revert if deadline has passed at supporting fee
  ✔ swap tokenA -> tokenB --> reapy tokenB at supporting fee (98ms)
  ✔ swap BNB -> token --> repay token at supporting fee (103ms)
  ✔ swap tokenA -> exact tokenB (89ms)
  ✔ swap tokenA -> full debt of tokenB (98ms)
  ✔ swap bnb -> exact tokenB (102ms)
  ✔ swap bnb -> full tokenB debt (152ms)
  ✔ Exact tokens -> BNB at supporting fee (82ms)
  ✔ Exact tokens -> BNB (66ms)
  ✔ Tokens -> Exact BNB (66ms)
  ✔ Tokens -> Exact BNB and supply
  ✔ Tokens -> full debt of BNB
Sweep Token
  ✔ Should be reverted if get zero address
  ✔ Sweep ERC-20 tokens (114ms)
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 (43ms)
    ✔ should properly set pending admin (40ms)
    ✔ should properly set pending admin twice (60ms)
    ✔ should emit event
  _acceptAdmin()
    ✔ should fail when pending admin is zero (39ms)
    ✔ should fail when called by another account (e.g. root) (62ms)
    ✔ should succeed and set admin and clear pending admin (64ms)
    ✔ should emit log on success (47ms)

Unitroller
  constructor
    ✔ sets admin to caller and addresses to 0 (41ms)
  _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 (46ms)
    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 (66ms)
    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 (64ms)
      ✔ set the correct fee
    setVAIMintCap(uint256)
      ✔ should revert if not authorised
      ✔ should set the correct mint cap
    setVenusTreasury(uint256)
      ✔ should revert if not authorised (67ms)
      ✔ 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 (52ms)
  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  (54ms)
      ✔ should revert if VAI transfer fails  (67ms)
      ✔ should revert if VAI to be burnt > vaiMinted  (54ms)
      should sucessfully perform the swap
        Fees: 10%
          ✔ stable token = 1$  (86ms)
          ✔ stable token < 1$  (124ms)
          ✔ stable token > 1$  (91ms)
        Fees: 0%
          ✔ stable token = 1$  (78ms)
          ✔ stable token < 1$  (79ms)
          ✔ stable token > 1$  (73ms)
      swapStableForVAI(address,uint256)
        ✔ should revert if receiver is zero address
        ✔ should revert if VAI mint cap will be reached  (103ms)
        ✔ should revert if amount after transfer is too small   (75ms)
        should sucessfully perform the swap
          Fees: 10%
            ✔ stable token = 1$  (97ms)
            ✔ stable token > 1$  (101ms)
            ✔ stable token < 1$  (102ms)
          Fees: 0%
            ✔ stable token = 1$  (88ms)
            ✔ stable token > 1$  (94ms)
            ✔ stable token < 1$  (95ms)
  PSM: 8 decimals
    initialization
      ✔ should revert if contract already deployed
      ✔ should initialize sucessfully
      reverts if init address = 0x0:
```

```
      ✔ acm
      ✔ treasury
      ✔ stableToken (54ms)
    reverts if fee init value is invalid
      ✔ feeIn
      ✔ feeOut
  Admin functions
    pause()
      ✔ should revert if not authorised (40ms)
      ✔ should pause if authorised (43ms)
      ✔ should revert if already paused (39ms)
    resume()
      ✔ should revert if not authorised (59ms)
      ✔ should resume if authorised (42ms)
      ✔ should revert if already resumed
    setFeeIn(uint256)
      ✔ should revert if not authorised (42ms)
      ✔ should revert if fee is invalid (42ms)
      ✔ set the correct fee (45ms)
    setFeeOut(uint256)
      ✔ should revert if not authorised (40ms)
      ✔ should revert if fee is invalid (72ms)
      ✔ set the correct fee (43ms)
    setVAIMintCap(uint256)
      ✔ should revert if not authorised (43ms)
      ✔ should set the correct mint cap (43ms)
    setVenusTreasury(uint256)
      ✔ should revert if not authorised (42ms)
      ✔ should revert if zero address (41ms)
      ✔ should set the treasury address (45ms)
    setOracle(address)
      ✔ should revert if not authorised (43ms)
      ✔ should revert if oracle address is zero (47ms)
      ✔ should set the oracle (59ms)
  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  (75ms)
      ✔ should revert if VAI transfer fails  (91ms)
      ✔ should revert if VAI to be burnt > vaiMinted  (74ms)
      should sucessfully perform the swap
        Fees: 10%
          ✔ stable token = 1$  (118ms)
          ✔ stable token < 1$  (118ms)
          ✔ stable token > 1$  (119ms)
        Fees: 0%
          ✔ stable token = 1$  (106ms)
          ✔ stable token < 1$  (103ms)
          ✔ stable token > 1$  (105ms)
      swapStableForVAI(address,uint256)
        ✔ should revert if receiver is zero address
        ✔ should revert if VAI mint cap will be reached  (108ms)
        should sucessfully perform the swap
          Fees: 10%
            ✔ stable token = 1$  (137ms)
            ✔ stable token > 1$  (137ms)
            ✔ stable token < 1$  (134ms)
          Fees: 0%
            ✔ stable token = 1$  (126ms)
            ✔ stable token > 1$  (124ms)
            ✔ stable token < 1$  (128ms)
  PSM: 6 decimals
    initialization
      ✔ should revert if contract already deployed (67ms)
      ✔ should initialize sucessfully
      reverts if init address = 0x0:
        ✔ acm
        ✔ treasury
        ✔ stableToken
```

```
    reverts if fee init value is invalid
      ✔ feeIn (41ms)
      ✔ feeOut
  Admin functions
    pause()
      ✔ should revert if not authorised (48ms)
      ✔ should pause if authorised (50ms)
      ✔ should revert if already paused (50ms)
    resume()
      ✔ should revert if not authorised (48ms)
      ✔ should resume if authorised (52ms)
      ✔ should revert if already resumed (50ms)
    setFeeIn(uint256)
      ✔ should revert if not authorised (49ms)
      ✔ should revert if fee is invalid (48ms)
      ✔ set the correct fee (51ms)
    setFeeOut(uint256)
      ✔ should revert if not authorised (47ms)
      ✔ should revert if fee is invalid (51ms)
      ✔ set the correct fee (51ms)
    setVAIMintCap(uint256)
      ✔ should revert if not authorised (54ms)
      ✔ should set the correct mint cap (54ms)
    setVenusTreasury(uint256)
      ✔ should revert if not authorised (49ms)
      ✔ should revert if zero address (53ms)
      ✔ should set the treasury address (59ms)
    setOracle(address)
      ✔ should revert if not authorised (51ms)
      ✔ should revert if oracle address is zero (54ms)
      ✔ should set the oracle (74ms)
  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  (87ms)
      ✔ should revert if VAI transfer fails  (105ms)
      ✔ should revert if VAI to be burnt > vaiMinted  (138ms)
      should sucessfully perform the swap
        Fees: 10%
          ✔ stable token = 1$  (139ms)
          ✔ stable token < 1$  (143ms)
          ✔ stable token > 1$  (142ms)
        Fees: 0%
          ✔ stable token = 1$  (127ms)
          ✔ stable token < 1$  (124ms)
          ✔ stable token > 1$  (130ms)
      swapStableForVAI(address,uint256)
        ✔ should revert if receiver is zero address
        ✔ should revert if VAI mint cap will be reached  (122ms)
        should sucessfully perform the swap
          Fees: 10%
            ✔ stable token = 1$  (160ms)
            ✔ stable token > 1$  (161ms)
            ✔ stable token < 1$  (164ms)
          Fees: 0%
            ✔ stable token = 1$  (146ms)
            ✔ stable token > 1$  (147ms)
            ✔ stable token < 1$  (146ms)

VAIController
  ✔ check wallet usdt balance (39ms)
  #getMintableVAI
    ✔ oracle
    ✔ getAssetsIn
    ✔ getAccountSnapshot
    ✔ getUnderlyingPrice (76ms)
    ✔ getComtroller (39ms)
    ✔ success (206ms)
  #mintVAI
```

```
    ✔ success (376ms)
  #repayVAI
    ✔ success for zero rate (231ms)
    ✔ success for 1.2 rate repay all (370ms)
    ✔ success for 1.2 rate repay half (336ms)
  #getHypotheticalAccountLiquidity
    ✔ success for zero rate 0.9 vusdt collateralFactor (372ms)
    ✔ success for 1.2 rate 0.9 vusdt collateralFactor (508ms)
  #liquidateVAI
    ✔ liquidationIncentiveMantissa
    ✔ success for zero rate 0.2 vusdt collateralFactor (1084ms)
    ✔ success for 1.2 rate 0.3 vusdt collateralFactor (1237ms)
  #getVAIRepayRate
    ✔ success for zero baseRate (39ms)
    ✔ success for baseRate 0.1 floatRate 0.1 vaiPirce 1e18 (229ms)
    ✔ success for baseRate 0.1 floatRate 0.1 vaiPirce 0.5 * 1e18 (244ms)
  #getVAIRepayAmount
    ✔ success for zero rate
    ✔ success for baseRate 0.1 floatRate 0.1 vaiPirce 1e18 (280ms)
    ✔ success for baseRate 0.1 floatRate 0.1 vaiPirce 0.5 * 1e18 (319ms)
  #getVAICalculateRepayAmount
    ✔ success for zero rate (58ms)
    ✔ success for baseRate 0.1 floatRate 0.1 vaiPirce 1e18 (418ms)
    ✔ success for baseRate 0.1 floatRate 0.1 vaiPirce 0.5 * 1e18 (456ms)
  #getMintableVAI
    ✔ include current interest when calculating mintable VAI (402ms)
  #accrueVAIInterest
    ✔ success for called once (177ms)
    ✔ success for called twice (263ms)
  #setBaseRate
    ✔ fails if access control does not allow the call (52ms)
    ✔ emits NewVAIBaseRate event (55ms)
    ✔ sets new base rate in storage (55ms)
  #setFloatRate
    ✔ fails if access control does not allow the call (50ms)
    ✔ emits NewVAIFloatRate event (53ms)
    ✔ sets new float rate in storage (53ms)
  #setMintCap
    ✔ fails if access control does not allow the call (50ms)
    ✔ emits NewVAIMintCap event (53ms)
    ✔ sets new mint cap in storage (55ms)
  #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 (41ms)
    ✔ sets ACM address in storage (41ms)

VAIVault
  ✔ claim reward (893ms)
  setVenusInfo
    ✔ fails if called by a non-admin
    ✔ fails if XVS address is zero
    ✔ fails if VAI address is zero
    ✔ disallows configuring tokens twice (70ms)

VRTVault
  unit tests
    setLastAccruingBlock
      ✔ fails if ACM disallows the call (53ms)
      ✔ fails if trying to set lastAccuringBlock to some absurdly high value (52ms)
      ✔ fails if lastAccuringBlock has passed (107ms)
      ✔ fails if trying to set lastAccuringBlock to some past block (55ms)
      ✔ fails if trying to set lastAccuringBlock to the current block (54ms)
      ✔ correctly sets lastAccuringBlock to some future block (80ms)
      ✔ can move lastAccuringBlock to a later block (132ms)
      ✔ can move lastAccuringBlock to an earlier block (128ms)
      ✔ fails if trying to move lastAccuringBlock to a block in the past (106ms)
```

```
    scenario
      ✔ deposit (182ms)
      ✔ should claim reward (101ms)
      ✔ should not claim reward after certain block (171ms)

  XVSVault
    setXvsStore
      ✔ fails if XVS is a zero address
      ✔ fails if XVSStore is a zero address
network block skew detected; skipping block events (emitted=3862 blockNumber13978)
network block skew detected; skipping block events (emitted=3862 blockNumber13978)
network block skew detected; skipping block events (emitted=3862 blockNumber13978)
      ✔ fails if the vault is already initialized
    add
      ✔ reverts if ACM does not allow the call (53ms)
      ✔ reverts if xvsStore is not set (56ms)
      ✔ reverts if a pool with this (staked token, reward token) combination already exists (73ms)
      ✔ reverts if staked token exists in another pool (53ms)
      ✔ reverts if reward token is a zero address (52ms)
      ✔ reverts if staked token is a zero address (53ms)
      ✔ reverts if alloc points parameter is zero (51ms)
      ✔ emits PoolAdded event (79ms)
      ✔ adds a second pool to an existing rewardToken (97ms)
      ✔ sets pool info (101ms)
      ✔ configures reward token in XVSStore (95ms)
    set
      ✔ reverts if ACM does not allow the call (53ms)
      ✔ reverts if pool is not found (54ms)
      ✔ reverts if total alloc points after the call is zero (75ms)
      ✔ succeeds if the pool alloc points is zero but total alloc points is nonzero (261ms)
      ✔ emits PoolUpdated event (78ms)
    setRewardAmountPerBlock
      ✔ reverts if ACM does not allow the call (54ms)
      ✔ reverts if the token is not configured in XVSStore (109ms)
      ✔ emits RewardAmountPerBlockUpdated event (100ms)
      ✔ updates reward amount per block (115ms)
    setWithdrawalLockingPeriod
      ✔ reverts if ACM does not allow the call (52ms)
      ✔ reverts if pool does not exist (54ms)
      ✔ reverts if the lock period is 0 (53ms)
      ✔ reverts if the lock period is absurdly high (53ms)
      ✔ emits WithdrawalLockingPeriodUpdated event (110ms)
      ✔ updates lock period (157ms)
    pendingReward
      ✔ includes the old withdrawal requests in the rewards computation (299ms)
      ✔ excludes the new withdrawal requests from the rewards computation (402ms)
    deposit
      ✔ reverts if the vault is paused (88ms)
      ✔ reverts if pool does not exist
      ✔ transfers pool token to the vault (133ms)
      ✔ updates user's balance (133ms)
      ✔ fails if there's a pre-upgrade withdrawal request (206ms)
      ✔ succeeds if the pre-upgrade withdrawal request has been executed (580ms)
      ✔ uses the safe _transferReward under the hood (346ms)
    executeWithdrawal
      ✔ fails if the vault is paused (89ms)
      ✔ only transfers the requested amount for post-upgrade requests (354ms)
      ✔ handles pre-upgrade withdrawal requests (375ms)
network block skew detected; skipping block events (emitted=14085 blockNumber15487)
network block skew detected; skipping block events (emitted=14085 blockNumber15487)
network block skew detected; skipping block events (emitted=14085 blockNumber15487)
      ✔ handles pre-upgrade and post-upgrade withdrawal requests (565ms)
    requestWithdrawal
      ✔ fails if the vault is paused (89ms)
      ✔ transfers rewards to the user (327ms)
      ✔ uses the safe _transferReward under the hood (339ms)
      ✔ fails if there's a pre-upgrade withdrawal request (184ms)
    claim
      ✔ fails if there's a pre-upgrade withdrawal request (95ms)
      ✔ succeeds if the pre-upgrade withdrawal request has been executed (348ms)
      ✔ excludes pending withdrawals from the user's shares (397ms)
      ✔ correctly accounts for updates in reward per block (267ms)
```

```
    ✔ uses the safe _transferReward under the hood (173ms)
  _transferReward
    ✔ sends the available funds to the user (148ms)
    ✔ emits VaultDebtUpdated event if vault debt is updated (110ms)
    ✔ does not emit VaultDebtUpdated event if vault debt is not updated (128ms)
    ✔ records the pending transfer (127ms)
    ✔ records several pending transfers (255ms)
    ✔ sends out the pending transfers in addition to reward if full amount <= funds available (440ms)
    ✔ sends a part of the pending transfers and reward if full amount > funds available (424ms)
  pendingWithdrawalsBeforeUpgrade
    ✔ returns zero if there were no pending withdrawals
    ✔ returns zero if there is only a new-style pending withdrawal (158ms)
    ✔ returns the requested amount if there is an old-style pending withdrawal (60ms)
    ✔ returns the total requested amount if there are multiple old-style pending withdrawals (101ms)
    ✔ returns zero if the pending withdrawal was executed (193ms)
  Scenarios
    ✔ works correctly with multiple claim, deposit, and withdrawal requests (1325ms)


  560 passing (3m)
```

Code Coverage

We recommend adding more tests to ensure the branch coverage is larger than 90% before deploying the contracts.

Fix Review: The client added additional tests to validate missing input validation fixes are implemented correctly. The client also added tests to confirm force liquidation functionality introduced during fix review.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
Comptroller/	100	90	100	100	
ComptrollerInterface.sol	100	100	100	100	
ComptrollerLensInterface.sol	100	100	100	100	
ComptrollerStorage.sol	100	100	100	100	
Unitroller.sol	100	90	100	100	
Comptroller/Diamond/	97.26	59.09	100	95.35	
Diamond.sol	97.26	59.09	100	95.35	109,228,229,230
Comptroller/Diamond/facets/	71.25	62	76.25	72.55	
FacetBase.sol	63.64	55.88	86.67	60.42	... 125,133,216
MarketFacet.sol	94.23	77.78	76.92	93.75	44,209,213,214
PolicyFacet.sol	67.86	62.96	72.22	68.97	... 368,369,371
RewardFacet.sol	0	0	0	0	... 181,184,185
SetterFacet.sol	86.96	77.78	95.45	87.14	... 491,492,509

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
XVSRewardsHelper.sol	94.12	75	100	95.45	78,107
Comptroller/Diamond/interfaces/	100	100	100	100	
IDiamondCut.sol	100	100	100	100	
IMarketFacet.sol	100	100	100	100	
IPolicyFacet.sol	100	100	100	100	
IRewardFacet.sol	100	100	100	100	
ISetterFacet.sol	100	100	100	100	
All files	76.51	62.6	81.37	77.51	

Changelog

- 2023-08-18 - Initial report
- 2023-09-18 - 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

or mean that Quantstamp endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. Quantstamp assumes no responsibility for the use of third-party software on any website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any output generated by such software.

Disclaimer

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

Quantstamp