OpenZeppelin

Venus Protocol Diamond Comptroller Audit

OPENZEPPELIN SECURITY | AUGUST 28, 2023

Security Audits

August 28, 2023

This security assessment was prepared by **OpenZeppelin**.

Table of Contents

Summary

Type	DeFi
Timeline	From 2023-07-07 To 2023-07-28
Languages	Solidity
Total Issues	16 (12 resolved, 1 partially resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	0 (0 resolved)
Low Severity Issues	6 (4 resolved)
Notes & Additional Information	10 (8 resolved, 1 partially resolved)

Scope

We audited the venus-protocol repository at the 94bc2e414e33ebf6c05d35c1605dcbd48fa932f5 commit.

In scope were the following contracts:

```
    ├── ComptrollerStorage.sol
    └── Diamond
        ├── Diamond.sol
        ├──facets
            ├──FacetBase.sol
            ├──MarketFacet.sol
            ├──PolicyFacet.sol
            ├──RewardFacet.sol
            ├──SetterFacet.sol
            └──XVSRewardsHelper.sol
        └── interfaces
            ├──IDiamondCut.sol
            ├──IMarketFacet.sol
            ├──IPolicyFacet.sol
            ├──IRewardFacet.sol
            └──ISetterFacet.sol
```

System Overview

The `Comptroller` is the core smart contract system responsible for managing markets and risk within the Venus lending protocol. It serves as a central hub and reliable source of truth for lending markets, carrying out crucial security measures and health checks for positions. This report focuses on a recent Comptroller update that restructures the contract to follow the diamond pattern.

In this update, the Comptroller's implementation has been adapted to a diamond pattern that maintains a mapping holding all function selectors and their corresponding facets. This structure allows the `Diamond` to route all function calls to their corresponding facets through a delegate call. The original storage remains untouched in the `Unitroller`, with the only new additions being variables related to the diamond pattern.

The Comptroller's functionality has been divided into the following facets:

the pool.

- The `PolicyFacet`, which oversees crucial security checks and risk management for positions. It houses functions consulted by markets to guarantee the health status of positions during key actions such as borrowing and liquidations.
- The `RewardFacet`, which takes charge of the computation and allocation of XVS rewards, ensuring their accurate distribution to qualifying addresses.
- The `SetterFacet`, which is mainly used by privileged roles to enable the modification of protocol configuration values.
- The `XVSRewardsHelper` facet, which contains internal functions used in `RewardFacet` and `PolicyFacet`.

Security Model and Trust Assumptions

Venus users are placing ultimate trust in the Comptroller `admin`, who can change critical safety parameters. The `admin` is currently set to the Venus governance contract.

Privileged Roles

The `admin` address can call multiple permissioned methods and enable custom access control for certain functions. The admin or admin-chosen addresses can perform the following actions:

- Add function selectors to the `Diamond`.
- Change the comptroller's implementation.
- Change the admin.
- Set custom access control for certain functions.
- Add new lending markets.
- Set the XVS accrual speed for any market.
- Give XVS grants.
- Call all setters in the setter facet to modify critical parameters such as the collateral factor or the oracle address.

Potential Irreversibility in `venusVAIVaultRate` Adjustments

If the `venusVAIVaultRate` is mistakenly changed to a very high number in `_setVenusVAIVaultRate()`, `releaseToVault()` will overflow.

If there is an attempt to correct the `venusVAIVaultRate`, `releaseToVault` will revert due to overflow, impeding any further change to the `venusVAIVaultRate`.

Consider adding input checks to prevent overflows.

***Update:** Acknowledged, not resolved. The Venus team stated:*

> *`venusVAIVaultRate` is set by the governance so the chances of setting it wrong are negligible. For now, we will just acknowledge the issue and no actions are needed from us.*

Incorrect Function Signature in `_setActionsPaused`

The `ensureAllowed` check in the `_setActionsPaused` function verifies whether a certain user is allowed to call the function by checking the `msg.sender` and function signature in the `AccessControlManager`.

However, the signature is calculated incorrectly. To calculate the 4-byte signature for `_setActionsPaused(address[] calldata markets_, Action[] calldata actions_, bool paused_)`, the canonical representation `_setActionsPaused(address[],uint8[],bool)` should be used instead of `_setActionsPaused(address[],uint256[],bool)`. `Actions` is an enum, and in Solidity `0.5.16` enums are mapped to the smallest `uint` type that is large enough to hold all the values. Since `Actions` holds 9 values, it will be mapped to `uint8`.

The correct function signature for the `_setActionsPaused` function should be `0x2b5d790c`.

In this case, since the canonical representation is used instead of the 4-byte signature when setting and checking the signature, the impact is relatively limited.

***Update:** Resolved in pull request #312 at commit cfaa69a.*

explanatory, the codebase could benefit from more complete NatSpec comments for all `public` and `external` functions. For instance:

- `getAssetsIn` in MarketFacet.sol
- `_setLiquidatorContract` in SetterFacet.sol
- `_setVAIMintRate` in SetterFacet.sol
- `_setTreasuryData` in SetterFacet.sol

Consider thoroughly documenting all functions (and their parameters) that are part of any contract's public API. Functions implementing sensitive functionality, even if not public, should be clearly documented as well.

Update: *Resolved in pull request #312 at commit 3909ff7.*

Implementation of EIP-2535 Does Not Fully Match the Specification

The `Diamond` contract implements the EIP-2535 standard, often referred to as the diamond proxy pattern.

The diamond proxy pattern is a proxy design where the functions are separated into multiple smaller 'facet' contracts. By breaking down the implementation contract into multiple facets, it is possible to build larger and more complex applications without exceeding the contract size limit.

However, there are some mismatches between the current implementation and the official specification worth highlighting:

- The diamondCut(IDiamondCut.FacetCut[] memory _diamondCut) function should be `diamondCut(IDiamondCut.FacetCut[] memory _diamondCut, address _init, bytes calldata _calldata)`.
- The `getFacetFunctionSelectors` function should be `facetFunctionSelectors`.
- The `getAllFacetAddresses` function should be `facetAddresses`.
- The `Diamond` contract does not implement the `facets` function.
- The `Diamond` contract does not implement the `facetAddress` function.

While the deviations from the specification may not be problematic for this particular use case, they may potentially cause errors in clients interacting with the `Diamond` contract who expect a fully-compliant implementation of EIP-2535. For example, tools such as Louper will not work as the function signatures of the functions used to inspect the `Diamond` do not match the ones from the standard.

Therefore, it is advisable to either modify the contract to make it fully compliant or clearly document the expected differences between the `Diamond` contract and the EIP.

Update: *Resolved in pull request #312 at commit 7417d8f.*

Possible Function Selector Clashing

Clashing can happen among functions with different names. Every function that is part of a contract's public ABI is identified, at the bytecode level, by a 4-byte identifier. This identifier depends on the function's signature, but since it is only 4 bytes, there is a possibility that two different functions with different function signatures may end up having the same identifier. The Solidity compiler tracks when this happens within the same contract, but not when the collision happens across different ones, such as between a proxy and its logic contract.

In this protocol, the `Unitroller` contract delegatecalls the `Diamond` contract which delegatecalls the facets. The `Unitroller` contract contains 8 `public`/`external` functions and the `Diamond` contract contains 46 `public`/`external` functions (this can increase in future upgrades).

The presence of these functions creates the possibility of a function selector clash. This can happen in the following scenarios:

- Functions in `Unitroller` and hardcoded functions in `Diamond` with the same function selector
- Functions in `Unitroller` and a facet with the same function selector
- Hardcoded functions in `Diamond` and a facet with the same function selector

Consider checking that no function selector collision is present when adding new functions to the `Diamond` (using `diamondCut`) or upgrading the `Diamond`'s implementation. Moving the hardcoded functions in `Diamond.sol` to a facet will also reduce the chances of a collision going unnoticed.

***Update:** Acknowledged, not resolved. The Venus team stated:*

> *We have just included the `diamondCut` functionality in the `Diamond.sol` file. For now, we will just acknowledge the issue, and no actions are needed from us.*

Unnecessary Access Allowance to the Comptroller Implementation

The security check `ensureAdminOr(comptrollerImplementation)` in `_setVenusSpeeds` and `_grantXVS` allows `msg.sender` to be the `admin` or the `comptrollerImplementation`. There is no reason to allow for `msg.sender == comptrollerImplementation` since the facets are called by the `Diamond` contract through `delegateCall`.

Allowing access from the `comptrollerImplementation` opens a potential attack path if the implementation were able to do calls to the `Unitroller`.

Consider disallowing access to these functions from the `ComtprollerImplementation`.

***Update:** Resolved in pull request #312 at commit 0aa7e17.*

Notes & Additional Information

Non-Explicit Imports

The use of non-explicit imports in the codebase can decrease the clarity of the code and may create naming conflicts between locally defined and imported variables. This is particularly relevant when multiple contracts exist within the same Solidity files or when inheritance chains are long.

Throughout the codebase, global imports are being used. Some instances are (but not limited to):

- Line 3 of `ComptrollerStorage.sol`

- Line 8 of `FacetBase.sol`
- Line 3 of `MarketFacet.sol`
- Line 4 of `MarketFacet.sol`
- Line 3 of `PolicyFacet.sol`

Following the principle that clearer code is better code, consider using named import syntax (*`import {A, B, C} from "X"`*) to explicitly declare which contracts are being imported.

Update: *Resolved in pull request #312 at commit 6d0a33c.*

Not Inheriting From Available Interfaces

The `Diamond` has multiple facets, and each facet has its own interface. However, the facet contracts are not explicitly inheriting their interfaces. This can lead to issues if an interface or corresponding contract is modified in a way that would make them incompatible.

Additionally, functions are added to the `Diamond` by calculating the function signatures from the interfaces instead of the contracts. Therefore it is important that the facet contracts explicitly inherit their respective interfaces to ensure the correct functions are added to the `Diamond`. For instance:

- `Diamond` does not inherit from `IDiamondCut`
- `MarketFacet` does not inherit from `IMarketFacet`
- `PolicyFacet` does not inherit from `IPolicyFacet`
- `RewardFacet` does not inherit from `IRewardFacet`
- `SetterFacet` does not inherit from `ISetterFacet`

To clarify intent, increase the readability of the codebase, and allow the compiler to perform more robust error-checking, consider updating the contracts' inheritance declarations to explicitly inherit from their corresponding interfaces.

Update: *Resolved in pull request #312 at commit 50761a0.*

Unnecessary Inheritances

inherits `ExponentialNoError`. Consider removing the explicit inheritance of `ExponentialNoError` in `MarketFacet`.

- `SetterFacet` inherits `ExponentialNoError`. However, `FacetBase` already inherits `ExponentialNoError`. Consider removing the explicit inheritance of `ExponentialNoError` in `SetterFacet`.

Inheriting a contract multiple times can be confusing and may lead to inconsistencies if the inherited contract is consuming storage slots. While the current version of `ExponentialNoError` is not consuming any storage slots, consider removing the duplicate inheritance to improve readability.

***Update:** Resolved in pull request #312 at commit 4c72e43.*

Lack of SPDX License Identifiers

Throughout the codebase, there are files that lack SPDX license identifiers. For instance:

- `ComptrollerStorage.sol`
- `Diamond.sol`
- `FacetBase.sol`
- `MarketFacet.sol`
- `PolicyFacet.sol`
- `RewardFacet.sol`
- `SetterFacet.sol`
- `XVSRewardsHelper.sol`
- `IDiamondCut.sol`
- `IMarketFacet.sol`
- `IPolicyFacet.sol`
- `IRewardFacet.sol`
- `ISetterFacet.sol`

To avoid legal issues regarding copyright and follow best practices, consider adding SPDX license identifiers to files as suggested by the Solidity documentation.

Throughout the codebase, several events do not have their parameters indexed. For instance:

- Line 12 and line 15 of `FacetBase.sol`
- Line 11, line 14, and line 17 of `MarketFacet.sol`
- Line 11 of `RewardFacet.sol`
- All events in `SetterFacet.sol`

Consider indexing event parameters to improve the ability of off-chain services to search and filter for specific events.

Update: *Resolved in pull request #312 at commit 5533343.*

Using `int` / `uint` Instead of `int256` / `uint256`

In the following contracts, there are instances where `int` / `uint` are used instead of `int256` / `uint256`:

- `ComptrollerStorage.sol`
- `FacetBase.sol`
- `MarketFacet.sol`
- `PolicyFacet.sol`
- `SetterFacet.sol`
- `IPolicyFacet.sol`
- `ISetterFacet.sol`

In favor of explicitness, consider replacing all instances of `int` / `uint` with `int256` / `uint256`.

Update: *Resolved in pull request #312 at commit 5533343.*

Local Variable Shares Name With Storage Variable

`venusAccrued` is declared as a local variable, but there is a storage variable with the same name.

Constants Not Using UPPER_CASE Format

In FacetBase.sol, there are several constants that are not using `UPPER_CASE` format. For instance:

- The `venusInitialIndex` constant declared on line 20
- The `closeFactorMinMantissa` constant declared on line 22
- The `closeFactorMaxMantissa` constant declared on line 24
- The `collateralFactorMaxMantissa` constant declared on line 26

According to the Solidity Style Guide, constants should be named with all capital letters with underscores separating words. For better readability, consider following this convention.

Update: *Acknowledged, not resolved. The Venus team stated:*

> *As we have dependencies on external contracts, if we change the convention the public variable `venusInitialIndex` and its getter will be changed. So for now we can't do the suggested change and will acknowledge the issue.*

Unnecessary Imports

Throughout the codebase, there are multiple instances of unnecessary imports that are either unused or already imported by other files.

- Import `ComptrollerStorage` of `Diamond.sol` which is already imported by `Unitroller`
- Import `ErrorReporter` of `FacetBase.sol` which is already imported by `VToken`
- Import `ErrorReporter` of `PolicyFacet.sol` which is already imported by `VToken`
- Import `ErrorReporter` of `SetterFacet.sol` which is already imported by `FacetBase`
- Import `PriceOracle` of `IMarketFacet.sol`
- Import `PriceOracle` of `IRewardFacet.sol`

Consider removing unused imports to improve the overall clarity and readability of the codebase.

There are general inconsistencies and deviations from the Solidity Style Guide throughout the codebase. Below is a non-exhaustive list of inconsistent coding styles.

While most external function names do not contain an underscore, some begin with one underscore. For example:

- `_supportMarket`
- `_setVenusSpeeds`
- `_grantXVS`

Some functions use named return variables, while others do not. For example:

- `getFacetFunctionSelectors` and `getAllFacetAddresses` declare named variables for the returned values.
- All the other functions in `Diamond.sol` do not declare a named variable for the return values.

Some facets are importing the `ComptrollerErrorReporter` contract while other facets are inheriting the `ComptrollerErrorReporter` contract. For example:

- FacetBase is importing the `ComptrollerErrorReporter` contract and therefore uses `ComptrollerErrorReporter.Error`.
- MarketFacet is inheriting the `ComptrollerErrorReporter` contract and therefore uses `Error`.

Consider enforcing a standard coding style, such as the one provided by the Solidity Style Guide, to improve the project's overall readability and consistency. Also, consider using a linter such as Solhint to define a style and analyze the codebase for style deviations.

***Update:** Partially resolved in pull request #312 at commit 4c72e43. The Venus team stated:*

> *We have not removed `_` from external methods as they are setters for the state variables and governance-controlled.*

The `Diamond` update refines the contract's structure and upgradeability mechanism, with negligible impact on function operations or underlying logic. This audit yielded 6 low-severity issues and 10 code quality notes.

Related Posts

Beefy Zap Audit

BeefyZapRouter serves as a versatile intermediary designed to execute users' orders through routes...

Security Audits

OpenBrush Contracts Library Security Review

OpenBrush is an open-source smart contract library written in the Rust programming language and the...

Security Audits

Linea Bridge Audit

Linea is a ZK-rollup deployed on top of Ethereum. It is designed to be EVM-compatible and aims to...

Security Audits

OpenZeppelin

Operation and Automation

Company

About us
Jobs
Blog

Contracts Library

Docs