OpenZeppelin | security

Venus Protocol Oracles Audit

June 6, 2023

This security assessment was prepared by OpenZeppelin.

Table of Contents

Table of Contents 2

Summary 3

Scope 4

System Overview 5

Design 5

Security Model and Trust Assumptions 6

Low Severity 8

L-01 Outdated Chainlink Interface 8

L-02 Misleading Documentation 8

Notes & Additional Information 9

N-01 Constants Not Using UPPER_CASE Format 9

N-02 Lack of Idiomatic Code Layout 9

N-03 Superfluous Casting 10

N-04 Global Namespace Pollution 10

N-05 Naming Suggestions 10

N-06 Ungraceful Handling of Function Call 11

N-07 Extraneous virtual and Visibility Keywords 12

N-08 Multiple Contract Declarations per File 13

N-09 Code Redundancy 13

N-10 Duplicated Event Parameter 14

N-11 Incomplete Event Emissions 14

N-12 Lack of Input Validation 15

N-13 Magic Constant 15

N-14 Unnecessary Repeated Fields in Structs 16

N-15 Unnecessary Variable Initialization in initialize 16

Conclusions 17

Appendix 18

Monitoring Recommendations 18

Summary

Type	DeFi
Timeline	From 2023-05-08 To 2023-05-23
Languages	Solidity

Total Issues	17 (8 resolved, 2 partially resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	0 (0 resolved)
Low Severity Issues	2 (2 resolved)
Notes & Additional Information	15 (6 resolved, 2 partially resolved)

Scope

We audited the `VenusProtocol/oracle` repository at the `78b1a41` commit.

```
contracts
├── interfaces
│   ├── FeedRegistryInterface.sol
│   ├── OracleInterface.sol
│   ├── PublicResolverInterface.sol
│   ├── PythInterface.sol
│   ├── SIDRegistryInterface.sol
│   └── VBep20Interface.sol
├── libraries
│   └── PancakeLibrary.sol
├── oracles
│   ├── BinanceOracle.sol
│   ├── BoundValidator.sol
│   ├── ChainlinkOracle.sol
│   ├── PythOracle.sol
│   └── TwapOracle.sol
└── ResilientOracle.sol
```

System Overview

The Venus protocol is a lending and borrowing solution on the Binance Smart Chain. As part of this protocol, oracles are required to obtain current fair market prices for certain assets. These oracles will be used for pricing core and isolated pools on Venus. The tokens that these oracles will be used for will be determined through governance. Any tokens that have more than 18 decimals, as well as fee-on-transfer and ERC-777 tokens will not be allowed.

Rather than relying on one single data feed, Venus has a `ResilientOracle` that obtains prices from various oracles, analyzes if these prices are trustworthy using the `BoundValidator` contract, and returns the resulting price. The specifics of how the `ResilientOracle` works are detailed below. The various other oracles used by this `ResilientOracle` are the `BinanceOracle`, `ChainlinkOracle`, `PythOracle`, and `TwapOracle`. Each of these oracle contracts designed by Venus calls their corresponding data feeds to obtain prices (e.g., `ChainlinkOracle` calls the oracle contract from Chainlink), and the `TwapOracle` uses the pool data from PancakeSwap.

The `ResilientOracle`, `BinanceOracle`, `BoundValidator`, `ChainlinkOracle`, `PythOracle`, and `TwapOracle` contracts are all meant to be used as implementation contracts and sit behind transparent proxies. This allows for upgradeability in the future. However, there are no plans to build more contracts that inherit from any of these oracles, nor the `BoundValidator`. Each of these contracts has a state variable named `_accessControlManager`, in which there is an address with default admin privileges that can determine which addresses are allowed to call the different functions in these oracles. In addition, all of these contracts inherit from OpenZeppelin's `Ownable2Step` contract, and the owner has the power to change the access control manager.

Design

As mentioned, the `ResilientOracle` contract obtains prices from other oracles and compares them to check whether a price is within certain boundaries, finally deciding whether the price is valid or not. These oracles are categorized as MAIN, PIVOT, and FALLBACK. For

someone to get a price of a listed asset, the `getUnderlyingPrice` function must be called, which works as follows:

- If the MAIN oracle is set, enabled, and returns a price that is not 0, and the PIVOT oracle is not enabled, then the MAIN oracle price is retrieved without checking it against any other oracle.
- If the MAIN oracle is set, enabled, and returns a price that is not 0, and the PIVOT price is enabled but returns an invalid price, then it checks that the MAIN price is within the boundaries defined in the `BoundValidator` contract with the FALLBACK price. If it is, it returns the MAIN price. Otherwise, it reverts.
- If the MAIN oracle is set, enabled, and returns a price that is not 0, and the PIVOT price is enabled and is not 0, it checks that the MAIN price is within the boundaries defined in the `BoundValidator` against the PIVOT price.
- If it is, it returns the MAIN oracle price.
- Otherwise, it checks that the FALLBACK price is within the boundaries of the PIVOT price.
    - If it is, it returns the FALLBACK price.
    - Otherwise, it checks that the MAIN price is within the boundaries of the FALLBACK price.
    - If it is, it returns the MAIN price.
    - Otherwise, it reverts.
- If the MAIN oracle is not set and the PIVOT price is set and is not 0, and the FALLBACK is set, is not 0, and is within the boundaries of the PIVOT price, then it returns the FALLBACK price. Otherwise, it reverts.

The MAIN, PIVOT, and FALLBACK oracles could be any of the oracle contracts mentioned above. Nevertheless, the `ResilientOracle` defines the `updateOracle` function, which assumes that the PIVOT contract will always be the `TwapOracle` by default.

Security Model and Trust Assumptions

The *owner* is a single address that can perform the critical administrative action of setting the access control manager. According to Venus, the owner is a time-locked contract that is used through the governance process. The owner is considered a trusted entity.

The *access control manager* is a contract, in which there is an address with default admin privileges that can perform the critical administrative action of giving and revoking roles for different addresses. These roles provide permissions for addresses to call privileged functions within the different oracle contracts as well as the `BoundValidator`. These functions include setting and enabling the oracle addresses, setting bounds configurations, adding or removing tokens from the oracles, setting data feed configurations, and even pausing and unpausing the contract. According to Venus, the address with the default admin privileges for the access control manager is a time-locked contract that is used through the governance process. Currently, this address is the same one as the owner.

The *data feeds* of Chainlink, Binance, and Pyth that are relied upon by the oracles are considered trusted. While ultimately the `ResilientOracle` returns the price to the user, if a malicious adversary were to be able to control one or more of these data feeds, they could implement a denial-of-service attack, or even worse, provide incorrect data to the system. The security of the system relies on the difficulty of taking control of one or more of these relatively centralized sources.

In addition, Venus will rely on PancakeSwap pools to provide the data feed for the `TwapOracle`. This source of data is decentralized and thus is more vulnerable to price manipulation, hence the requirement for TWAP. The difficulty of price manipulations for these pools is directly related to their liquidity. Therefore, the security of this oracle model is dependent on using tokens that are more popular and have a larger pool depth. With the decline in popularity of PancakeSwap V2 and the migration to V3, it will be important for the protocol to closely monitor the tokens of interest.

Lastly, it is worth noting that all the mentioned oracle contracts feature a `getOraclePrice` function. However, regardless of the number of decimals associated with the underlying asset or the vToken address, these functions consistently return prices with 36 decimals. This applies to the `ResilientOracle` as well, which serves as the primary point of interaction for other contracts. It is assumed that this consideration is taken into account throughout the out-of-scope codebase when invoking the `getOraclePrice` functions and performing calculations based on the returned results.

Low Severity

L-01 Outdated Chainlink Interface

In line 201 of `ChainlinkOracle`, the interface `AggregatorV2V3Interface` is used, which inherits from both the `AggregatorInterface` and the `AggregatorV3Interface`. Chainlink recommends using the `AggregatorV3Interface`, as shown in its documentation. This prevents the usage of deprecated functions in the `AggregatorInterface`, which do not throw errors if no answer has been reached, but instead return 0. The dependence on this unexpected behavior increases the attack surface for the calling contract.

Consider updating the interface used in the `ChainlinkOracle` from `AggregatorV2V3Interface` to `AggregatorV3Interface`.

Update: *Resolved in pull request #84 at commit ddd4b02.*

L-02 Misleading Documentation

The following documentation is misleading:

- Line 83 of `BoundValidator` says "range error thrown if lower bound is greater than upper bound" instead of "range error thrown if lower bound is greater than or equal to upper bound".
- Line 18 of `TwapOracle` says that the `baseUnit` signifies the "decimals of the underlying asset", but it actually takes the value of `1e{decimals}` of the asset.
- Line 25 of `TwapOracle` says that "XVS-WBNB is not reversed, while WBNB-XVS is", when in reality the opposite is true.

Consider updating these misleading comments to match the code's intention.

Update: *Resolved in pull request #84 at commit f4352f1.*

Notes & Additional Information

N-01 Constants Not Using `UPPER_CASE` Format

In `TwapOracle.sol` the following constants are not using `UPPER_CASE` format:

- The `bnbBaseUnit` constant declared on line 47
- The `busdBaseUnit` constant declared on line 48

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: *Resolved in pull request #84 at commit 70a2211.*

N-02 Lack of Idiomatic Code Layout

According to the Solidity Style Guide, elements inside of contracts, libraries, and interfaces should be in the following order: type declarations, state variables, events, errors, modifiers, and finally functions.

In addition, the guide states that "functions should be grouped according to their visibility and ordered" first by the constructor, followed by the receive function (if it exists), the fallback function (if it exists), public functions, internal functions, and finally private functions. In addition, since the contracts are implementation contracts, it is idiomatic to place the `initialize` function directly below the constructor.

Throughout the codebase, the layouts of the files do not follow the Solidity Style Guide and are not consistent with each other. Consider moving the modifiers, events, errors, and functions in these files to match the Solidity Style Guide.

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

> *Ignoring the Pyth Interface because it was copied from the original project and linting would make the diff more complicated when updating.*

N-03 Superfluous Casting

In line 305 and line 307 of `TwapOracle`, the parameter `vToken`, which is already of type `address`, is unnecessarily cast with the `address` keyword.

In addition, in lines 182 and 207, the value 18 is unnecessarily cast with the `uint256` keyword.

Consider removing the superfluous casting in order to improve the readability of the codebase.

Update: *Resolved in pull request #84 at commit 66707bd.*

N-04 Global Namespace Pollution

In `BoundValidator`, `ChainlinkOracle`, `PythOracle`, and `TwapOracle`, there are structs defined outside of the contracts and therefore in the global namespace. This pattern is used when there are multiple contract implementations within one source file that need to reference the same structs. However, in these files in the codebase, there is only one implementation contract per file, so this methodology is unnecessary.

Consider either placing these structs inside the contracts' definitions or within the corresponding interfaces to reduce global namespace pollution.

Update: *Resolved in pull request #84 at commit 28f4924.*

N-05 Naming Suggestions

To favor explicitness and readability, the following locations in the contracts may benefit from better naming:

- The interface file names as well as the interfaces themselves could take the format of `I{...}` as opposed to `{...}Interface`. For example, `VBep20Interface.sol` could be named `IVBep20.sol`.
- In `ResilientOracle`:
  - The `struct` field `enableFlagsForOracles` could be renamed to `isEnabled`.
  - The function `enableOracle` could be changed to `updateOracleEnablement` to reflect both the enablement and disablement of the oracle.

- The function `updatePrice` could be changed to `updatePivotPrice` to provide clarity as to which oracle is being updated.
- The state variable `tokenConfigs` could be changed to `_tokenConfigs` since it is a `private` variable.

- In `BinanceOracle`:
  - The variable `decimalDelta` could be renamed to `decimals` to avoid confusion with other parts of the codebase, as `decimalDelta` typically refers to the value `18 - decimals`.
  - The function `compare` could be renamed to `_compare` since it is an `internal` function.

- In `BoundValidator`:
  - The struct `ValidateConfig` could be renamed to `ValidatorConfig` or `BoundValidatorConfig` for consistency.

- In `TwapOracle`:
  - The `struct` field `isBnbBased` could be renamed to `isWbnbBased`.
  - The function `pokeWindowValues` could be renamed to `_pokeWindowValues` since it is an `internal` function.

Consider renaming as suggested above to improve the consistency and readability of the codebase.

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

> *We prefer to reduce the number of changes.*

N-06 Ungraceful Handling of Function Call

When the function `_getUnderlyingAsset` in `ResilientOracle` is called, if the passed `vToken` does not equal that of `vBnb` nor `vai`, it calls the function `VBep20Interface(vToken).underlying()`. If the address passed in as `vToken` does not have the `underlying` method, it does not fail gracefully with an error message.

Similarly, there are instances of ungraceful handling in `getUnderlyingPrice` in `BinanceOracle`, `_getUnderlyingAsset` in `BoundValidator`, `setUnderlyingPrice` and `_getUnderlyingPriceInternal` in `ChainlinkOracle`, `getUnderlyingPrice` in `PythOracle`, and `_getUnderlyingAsset` in `TwapOracle`.

Consider handling these cases gracefully using `try-catch` and returning descriptive error messages.

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

> *We prefer to reduce the number of changes.*

N-07 Extraneous `virtual` and Visibility Keywords

Throughout the codebase, there are extraneous `virtual` and visibility keywords. From what is understood, there are no plans to inherit from existing contracts. Thus, it is best practice to limit the scope to what is designed, thereby removing extraneous `virtual` keywords and using more restrictive visibility keywords.

Places with unnecessary `virtual` keywords:

- In `BoundValidator`, in the functions `setValidateConfigs`, `setValidateConfig`, and `validatePriceWithAnchorPrice`
- In `TwapOracle`, in the function `_updateTwapInternal`

Places with unnecessary `internal` keywords (replace with `private`):

- In `ResilientOracle`, in the functions `_getMainOraclePrice`, `_getFallbackOraclePrice` and `_getUnderlyingAsset`
- In `BinanceOracle`, in the function `compare`
- In `BoundValidator`, in the functions `_isWithinAnchor` and `_getUnderlyingAsset`
- In `ChainlinkOracle`, in the functions `_getUnderlyingPriceInternal` and `_getChainlinkPrice`
- In `TwapOracle`, in the functions `_updateTwapInternal`, `pokeWindowValues`, and `_getUnderlyingAsset`

Places with unnecessary `public` keywords (replace with `external`):

- In `ResilientOracle`, in the function `initialize`
- In `BinanceOracle`, in the functions `initialize` and `getUnderlyingPrice`
- In `BoundValidator`, in the functions `initialize` and `validatePriceWithAnchorPrice`
- In `ChainlinkOracle`, in the functions `initialize` and `getUnderlyingPrice`
- In `PythOracle`, in the functions `initialize` and `getUnderlyingPrice`
- In `TwapOracle`, in the functions `initialize` and `updateTwap`

Consider removing extraneous `virtual` keywords and using more restrictive visibility keywords following the recommendations above.

Update: *Resolved in pull request #84 at commit e11f135.*

N-08 Multiple Contract Declarations per File

In the file `PythInterface`, there is a contract `PythStructs`, an interface `IPyth`, and an abstract contract `AbstractPyth`. To improve understandability and readability for developers and reviewers, it is recommended to include one contract or interface per file.

The contract `PythStructs` contains no functions, and only two structs. Consider moving these structs into `PythOracle`. Consider removing `AbstractPyth` completely, since no contracts inherit it.

The file `PancakeLibrary` also contains more than one library and interface declaration in the same file. Consider separating the interface and libraries into their own files to improve the readability of the codebase.

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

> *We prefer to reduce the number of changes.*

N-09 Code Redundancy

We recommend the following high-level refactors to improve the readability, composability and overall quality of the codebase:

- In the `ResilientOracle` contract, the `_getFallbackOraclePrice` and `_getMainOraclePrice` functions have very similar code. They can be consolidated into one function to prevent changes in two different places, which increases the space for error in future changes to the codebase.
- The `_getUnderlyingAsset` function is very similar or the same in many contracts, such as `ResilientOracle` and `BoundValidator`. Consider moving this functionality to a separate library.
- Similarly, the `setTokenConfigs` and `setTokenConfig` functions in the `ResilientOracle`, `ChainlinkOracle`, `TwapOracle`, and `PythOracle` are practically the same and can be moved to a library.
- The `setDirectPrice` and `setUnderlyingPrice` functions in the `ChainlinkOracle` contract share almost all their functionality. The

`setDirectPrice` function could be called from the `setUnderlyingPrice` function after looking up the `asset` address from the `vToken` address, or alternatively an internal `_setPrice(asset)` function that is called from both functions could be introduced.

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

> *We prefer to reduce the number of changes.*

N-10 Duplicated Event Parameter

In the `ChainlinkOracle` contract, the `PricePosted` event defined has four fields. This event is only used in line 80 and line 95. Since in both cases the last two fields (`requestedPriceMantissa` and `newPriceMantissa`) are always equal, consider removing one of them.

Update: *Resolved in pull request #84 at commit 5d811fd.*

N-11 Incomplete Event Emissions

The following suggestions are provided to improve the ability of off-chain services to search and filter for specific events:

- The `PythOracleSet` event in the `PythOracle` contract should emit both the old and new oracle addresses. In the `initializer` function, it should emit `(0, underlyintPythOracle_)` and in the `setUnderlyingPythOracle` function it should emit (old, new), similar to the `PricePosted` event of the `ChainlinkOracle` contract, which emits both the old and new oracle price.
- The `BatchPriceFeedUpdate` event in `PythInterface` does not have any parameters indexed. The `chainId` field could be indexed.

Update: *Partially resolved in pull request #84 at commit cf6a66c. Regarding the first bullet point, the Venus team updated the `PythOracleSet` event to emit both the old and new oracle addresses. Furthermore, the `initializer` and `setUnderlyingPythOracle` functions, which emit this event, have been updated correspondingly. Regarding the second bullet point, the Venus team stated:*

> *Ignoring the Pyth Interface because it was copied from the original project and linting would make the diff more complicated when updating.*

N-12 Lack of Input Validation

We recommend implementing the following input validations:

- In the `setOracle` function of the `ResilientOracle` contract, it is advisable to check that for non-`MAIN` oracles (i.e., `PIVOT` and `FALLBACK`), the specified address is different from the `MAIN` oracle address.
- In the `setTokenConfig` function of the `ResilientOracle` contract, there are currently no checks in place to ensure that a new configuration does not override an existing one. To address this, it is recommended to verify that the existing token asset address in the `tokenConfigs` is a zero address. By implementing this validation, updating the oracle for a specific token would only be possible through the `setOracle` and `enableOracle` functions, reducing the likelihood of errors.

While this issue does not pose a security risk, the absence of validation on user-controlled parameters may lead to erroneous transactions, particularly if some clients default to sending null parameters when none are specified.

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

> *These functions will be executed via Governance, with a timelock of 3 days, and the vote of the community, so there should not be errors in the inputs.*

N-13 Magic Constant

The usage of the constant `100e16` on line 123 of `BoundValidator.sol` lacks sufficient explanation regarding its origin and purpose.

Developers should define a constant variable for every magic value used (including booleans), giving it a clear and self-explanatory name. Additionally, for complex values, inline comments explaining how they were calculated or why they were chosen are highly recommended.

Consider following the Solidity Style Guide to define the constant in `UPPER_CASE_WITH_UNDERSCORES` format with the corresponding specific public getter to read it.

Update: *Partially resolved in pull request #84 at commit c420a18. The Venus team stated:*

> *Mitigated. We added a comment to explain why the multiplication by the constant `1e18` is needed.*

N-14 Unnecessary Repeated Fields in Structs

The contracts `ResilientOracle`, `TwapOracle`, `PythOracle`, and `ChainlinkOracle` define a `TokenConfig` struct. This struct is utilized in the `tokenConfigs` mappings to track various properties of each token's configuration. However, these mappings already use the asset address as the key for each asset, while also including an asset field within the structs. This repetition results in storing unnecessary information.

A similar pattern can be observed in the `ValidateConfig` struct within the `BoundValidator` contract.

To optimize storage and avoid redundant data, it is recommended to remove the asset field from the structs. Instead, consider utilizing an existing field within the struct or introducing a boolean field to check the configuration's validity.

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

> *We prefer to reduce the number of changes.*

N-15 Unnecessary Variable Initialization in `initialize`

The `boundValidator` variable in the `ResilientOracle` contract is currently set in the `initialize` function. However, since this variable remains unchanged after initialization, it can be optimized by declaring it as an immutable variable in the constructor.

By making `boundValidator` immutable, initialization inconsistencies can be avoided, and gas costs can be reduced when accessing it. This is especially beneficial when calling the `getUnderlyingPrice` function, where accessing `boundValidator` may occur up to three times.

To improve efficiency and maintain consistency, consider declaring `boundValidator` as an immutable variable in the constructor of the `ResilientOracle` contract.

Update: *Resolved in pull request #84 at commit 4f56661.*

Conclusions

An in-depth review of the oracles was performed, and several low-severity issues and notes were identified in the system. The Venus Protocol team provided dedicated documentation, which clarified the system's design. The Venus team was very responsive throughout the audit.

Appendix

Monitoring Recommendations

While audits help in identifying code-level issues in the current implementation and potentially the code deployed in production, the Venus Protocol team is encouraged to consider incorporating monitoring activities in the production environment. Ongoing monitoring of deployed contracts helps identify potential threats and issues affecting production environments. With the goal of providing a complete security assessment, the monitoring recommendations section raises several actions addressing trust assumptions and out-of-scope components that can benefit from on-chain monitoring. Consider monitoring the following items:

- Commonly reverting oracles may indicate a lack of fresh data from their underlying feeds.
- Frequent failing of the bound validator may indicate that the TWAP oracle has low volume or liquidity.
- The amount of liquidity in the pools that are used by the TWAP oracle.
- Any change of ownership or granting of privileged roles.