Venus - Cached prices and Capped oracles

Security Assessment

CertiK Assessed on Apr 30th, 2025

CertiK Assessed on Apr 30th, 2025

Venus - Cached prices and Capped oracles

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES
DEX, Oracle

ECOSYSTEM
Binance Smart Chain (BSC)

METHODS
Formal Verification, Manual Review, Static Analysis

LANGUAGE
Solidity

TIMELINE
Delivered on 04/30/2025

KEY COMPONENTS
N/A

CODEBASE
https://github.com/VenusProtocol/oracle/
View All in Codebase Page

COMMITS
17be0ef74817da3c9163f886d6e3d9aa6bd0c7ce
8c2e6d33585ab29c71a359c48054121b369ba1b3
a0bcec7048e1fec1a7514a8cfe97c7968b915fbe
View All in Codebase Page

Vulnerability Summary

17	15	1	0	1	0
Total Findings	Resolved	Timelock	Partially Resolved	Acknowledged	Declined

1 Centralization	1 Timelock		Centralization findings highlight privileged roles & functions and their capabilities, or instances where the project takes custody of users' assets.
1 Critical	1 Resolved		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
0 Major			Major risks may include logical errors that, under specific circumstances, could result in fund losses or loss of project control.
2 Medium	2 Resolved		Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
4 Minor	4 Resolved		Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
9 Informational	8 Resolved, 1 Acknowledged		Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS

VENUS - CACHED PRICES AND CAPPED ORACLES

CODEBASE | VENUS - CACHED PRICES AND CAPPED ORACLES

Repository

https://github.com/VenusProtocol/oracle/

Commit

17be0ef74817da3c9163f886d6e3d9aa6bd0c7ce 8c2e6d33585ab29c71a359c48054121b369ba1b3
a0bcec7048e1fec1a7514a8cfe97c7968b915fbe

AUDIT SCOPE | VENUS - CACHED PRICES AND CAPPED ORACLES

39 files audited ● 1 file with Acknowledged findings ● 31 files with Resolved findings ● 7 files without findings

ID	Repo	File	SHA256 Checksum
CTO	VenusProtocol/oracle	oracles/common/CorrelatedTokenOracle.sol	e617e8f7d0e139667e20f015e34fb2b3b4ae347ba37270c467b08a57bac547ce
ROV	VenusProtocol/oracle	ResilientOracle.sol	bc4186a4a1ccd13fd38dbc773b4d2d8c63bfc5e4a5da2eaa6b4eddab6480c615
TVP	VenusProtocol/oracle	lib/Transient.sol	fbca587773815d1a0cffca42c9d38c5961164fafb3053662ad72141f1edb930c
ABN	VenusProtocol/oracle	oracles/AnkrBNBOracle.sol	227c70ec043bca57deafcbcba23d0f725b4170c33b4e44d9f7adffc2ad0466b0
BNB	VenusProtocol/oracle	oracles/BNBxOracle.sol	febbc4c863f9147601c87b81c179335e3912de802c0e71b5e4135253861f00ff
ERC	VenusProtocol/oracle	oracles/ERC4626Oracle.sol	7e9c1a8246c658a82f4efcb87af52de271f0b4d1c6fb69ba529261941a931f10
EAO	VenusProtocol/oracle	oracles/EtherfiAccountantOracle.sol	d07d10c280ad024d236a6d398742f6025722d4735b92c7aaeb062c6be750fb09
OJO	VenusProtocol/oracle	oracles/OneJumpOracle.sol	19c29d159864d237cfe311008aae609682d0f2d1b81297cdfbeb6effea791489
POV	VenusProtocol/oracle	oracles/PendleOracle.sol	29338d5722877901f8a8f22fc1010c554e01f39e875d2b4d40ed8aa67a3a1987
SFO	VenusProtocol/oracle	oracles/SFraxOracle.sol	bb52cc39e988153b240e07ee9538c0d42c28d45d9c0bce59d23e69120cbf007d
SBN	VenusProtocol/oracle	oracles/SlisBNBOracle.sol	f633a31945639bf7d3cab5bf01c912cf6494d4b3eadbe3dc65639511c18d9569
SBB	VenusProtocol/oracle	oracles/StkBNBOracle.sol	1a6bc5d4021c121781fb32216be9f9cfa36a364a9d42049275f43c2312d51df9
WBE	VenusProtocol/oracle	oracles/WBETHOracle.sol	0a367d397dbd8964131aca720d7722c6c785fcf6900717063e7504addd38731f
WET	VenusProtocol/oracle	oracles/WeETHAccountantOracle.sol	abd9c7edb4e128fec7fe2f7dc0abef5b0826c165e4f6f86871f7fd9851fd696d

ID	Repo	File	SHA256 Checksum
WEH	VenusProtocol/oracle	oracles/WeETHOracle.sol	daf53a6c658e44c3db219afa979d7c8cb8beafa74f37afc1560c2ad24e5be5b7
WEO	VenusProtocol/oracle	oracles/WstETHOracleV2.sol	ff07608d67a23c5cbefe6c5f55fae7310b4d29d8cd7904bdb3c3fc4cf1e91e37
CTV	VenusProtocol/oracle	oracles/common/CorrelatedTokenOracle.sol	1a9a677e6b9616e8d645a9552c3dd9bafa8a0bbe676a2c55e49dc075e02f7ed6
ABB	VenusProtocol/oracle	oracles/AnkrBNBOracle.sol	bad2f6b796f97a80295d9aa02206a4434de3b6702aa628c5d8c746f962adfb17
ABO	VenusProtocol/oracle	oracles/AsBNBOracle.sol	21da84f707e0b86495ae9ce11751a89bd88b2d7d549400550a355bddf2fb7380
BNO	VenusProtocol/oracle	oracles/BNBxOracle.sol	5118d09cde0e5b802dafbeb2a72db206ea9cc4364fd6df42a09afc5f621a1c1a
ERO	VenusProtocol/oracle	oracles/ERC4626Oracle.sol	d8a81b91939e73ff3b92802e75668495fe45c0ccaacccf92c61c12fce59e8e8d
EAV	VenusProtocol/oracle	oracles/EtherfiAccountantOracle.sol	7965bab35486bf492d18456356954d34c93539b65fb9e25f043b581c7f14c05c
OJV	VenusProtocol/oracle	oracles/OneJumpOracle.sol	5f407e8cd5d1a50c8ac80e7d41d94b9d82d934a88626699cfb32e6d7a01e273a
SFV	VenusProtocol/oracle	oracles/SFraxOracle.sol	e175a96bd7600190472506303a523d56894aa048a77a0d90d4561180b4342c4a
SBO	VenusProtocol/oracle	oracles/SlisBNBOracle.sol	d46591cdfd6b1ecb8434dfc231c0472543613f5fc6df5f5d70b3e3d47c543c20
SBV	VenusProtocol/oracle	oracles/StkBNBOracle.sol	46e1070670b399294541377ab2a34eafd5d69b78daaedc30eed22e4a27ae142e
WBT	VenusProtocol/oracle	oracles/WBETHOracle.sol	e1205726fa246bf3e7c358800ec71fb94afd2e44265d1b5c6c829a409d025844
WEA	VenusProtocol/oracle	oracles/WeETHAccountantOracle.sol	9671319b6ad5453066a28c5e8eae93ec7af64614118f6998298d26c8943b8062
WEV	VenusProtocol/oracle	oracles/WeETHOracle.sol	a27276980aa506da3ac6819669478fe5f9fafdce77a8f9fb700baff0399e4be5
WEP	VenusProtocol/oracle	oracles/WstETHOracleV2.sol	d2f1fd1c995f7b9639822e85b06211383a506c90db9fc06d5cd352333bfc77c1
ZET	VenusProtocol/oracle	oracles/ZkETHOracle.sol	c74d42c1c5a0ec243225308cc0899c7194463e0038c3d83754a3b48b5cb05bf0

ID	Repo	File	SHA256 Checksum
ROP	VenusProtocol/oracle	ResilientOracle.sol	1476df1cd0622bd91602c2157629d6cef0fd61b73735454225f1ec373b50d932
ICO	VenusProtocol/oracle	interfaces/ICappedOracle.sol	2a81bb711bebb55769d0a4a20246067b7932f4fcf54869c5a65f815ff53ed11f
TRA	VenusProtocol/oracle	lib/Transient.sol	fbca587773815d1a0cffca42c9d38c5961164fafb3053662ad72141f1edb930c
BVV	VenusProtocol/oracle	oracles/BoundValidator.sol	6605137a92d35105cdac2123ad3389bab28abb0423ca7a6da117f8a3412c0ade
COV	VenusProtocol/oracle	oracles/ChainlinkOracle.sol	2d48c52e9eff8507a6a8843a52a6b22d903bf82fb3dd2f76fc1da370affeec3d
POP	VenusProtocol/oracle	oracles/PendleOracle.sol	2ed513f5bf24da772b178a292e6f69b35fc3d5c519c09f7e91224dc7c2e37198
ICV	VenusProtocol/oracle	interfaces/ICappedOracle.sol	2a81bb711bebb55769d0a4a20246067b7932f4fcf54869c5a65f815ff53ed11f
OIV	VenusProtocol/oracle	interfaces/OracleInterface.sol	271e6bdb34dd65d9cc611b7c2215e7fe39d7a4a08080dc440e83bb57c568062d

APPROACH & METHODS

VENUS - CACHED PRICES AND CAPPED ORACLES

This report has been prepared for Venus to discover issues and vulnerabilities in the source code of the Venus - Cached prices and Capped oracles project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

SUMMARY | VENUS - CACHED PRICES AND CAPPED ORACLES

This audit concerns the changes made in files outlined in:

- PR-239

Note that any centralization risks present in the existing codebase before these PRs were not considered in this audit and only those added in these PRs are addressed in the audit. We recommend all users carefully review the centralization risks, much of which can be found in our previous audits, which can be found here: https://skynet.certik.com/projects/venus.

The purpose of the PR is to use transient storage to cache exchange rates and prices during calls to `updatePrice()` / `updateAssetPrice()` to be used when `_getPrice()` is called and reduce gas costs of continually reading the price from oracles. In addition, it adds a cap to the exchange rate returned by the `CorrelatedTokenOracle`.

In updates to the code, the caching of exchange rates was removed and a `snapshotGap` was added. The `snapshotGap` adds a gap between the maximum exchange rate curve for a snapshot interval and the current exchange rates, allowing for better handling of short term fluctuations.

DEPENDENCIES

VENUS - CACHED PRICES AND CAPPED ORACLES

Third Party Dependencies

The protocol is serving as the underlying entity to interact with third party protocols. The third parties that the contracts interact with are:

- Oracles
- Correlated Token Contracts (See respective oracles for the specific contracts.)
- Underlying Token Contracts (See respective oracles for the specific contracts.)

The scope of the audit treats third party entities as black boxes and assumes their functional correctness. However, in the real world, third parties can be compromised and this may lead to lost or stolen assets. Moreover, updates to the state of a project contract that are dependent on the read of the state of external third party contracts may make the project vulnerable to read-only reentrancy. In addition, upgrades of third parties can possibly create severe impacts, such as increasing fees of third parties, migrating to new LP pools, etc.

Assumptions

Within the scope of the audit, assumptions are made about the intended behavior of the protocol in order to inspect consequences based on those behaviors. Assumptions made within the scope of this audit include:

- We assume that each supported assets will be properly vetted and have the appropriate configurations made. In particular, we assume each supported asset will be vetted for potential inflation attacks and either not supported or have proper configurations and mitigations set to ensure the protocols saftey.

Recommendations

We recommend constantly monitoring the third parties involved to mitigate any side effects that may occur when unexpected changes are introduced, as well as vetting any third party contracts used to ensure no external calls can be made before updates to its state.

FINDINGS | VENUS - CACHED PRICES AND CAPPED ORACLES

17	1	1	0	2	4	9
Total Findings	Critical	Centralization	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Venus - Cached prices and Capped oracles. Through this audit, we have uncovered 17 issues ranging from different severity levels. Utilizing the techniques of Static Analysis & Manual Review to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
ERC-01	`ERC4626Oracle` Will Always Return A Price Of Zero As `ONE_CORRELATED_TOKEN` Is Not Initialized	Logical Issue	Critical	Resolved
VCP-04	**Centralization Related Risks**	**Centralization**	**Centralization**	**48h Timelock**
CTO-08	Issues With Exchange Rate Manipulations	Logical Issue	Medium	Resolved
VPB-01	Differing Behavior If Oracles Price Differs During Transaction	Logical Issue	Medium	Resolved
CTO-02	Missing Input Validation	Logical Issue	Minor	Resolved
CTO-03	`isCapped()` May Return True When It Is Not Capped	Logical Issue	Minor	Resolved
CTO-04	Issue If `maxAllowedExchangeRate = 0`	Logical Issue	Minor	Resolved
VCP-01	Discussion On `ACCESS_CONTROL_MANAGER`	Inconsistency	Minor	Resolved
CTO-05	Event Not Indexed	Design Issue	Informational	Resolved
CTO-06	Initial Values Should Be Verified Before Deployment	Logical Issue	Informational	Acknowledged

ID	Title	Category	Severity	Status
CTO-07	Contract Doesn't Inherit Interface	Logical Issue	Informational	Resolved
VCP-02	Local Variable Shadowing	Coding Style	Informational	Resolved
VCP-03	Specific Imports Are Not Used	Coding Style	Informational	Resolved
VCP-05	Misleading Name	Inconsistency	Informational	Resolved
VPB-02	`CACHE_SLOT` Does Not Follow ERC-7201	Logical Issue	Informational	Resolved
VPB-03	Missing/Incomplete Natspec	Inconsistency	Informational	Resolved
VPB-04	Typos And Inconsistencies	Inconsistency	Informational	Resolved

ERC-01 `ERC4626Oracle` WILL ALWAYS RETURN A PRICE OF ZERO AS `ONE_CORRELATED_TOKEN` IS NOT INITIALIZED

Category	Severity	Location	Status
Logical Issue	Critical	oracles/ERC4626Oracle.sol (Base): 13	Resolved

Description

In the contract `ERC4626Oracle`, `ONE_CORRELATED_TOKEN` is not initialized and will be set to the default value of 0. As such, when `_getUnderlyingAmount()` is called it will return 0, so that `getPrice()` will always return a value of 0.

If it is used as a standalone oracle, it can return a price of 0 significantly devaluing the asset. If it is used as an oracle within the `ResilientOracle`, then it can cause a denial of service as it will revert if a price of 0 is returned.

Recommendation

We recommend initializing `ONE_CORRELATED_TOKEN` to be `10 ** IERC4626(correlatedToken).decimals()`.

Alleviation

`[CertiK, 03/06/2025]`: The client made the recommended changes resolving the finding in commit b682c4427368c8e34ddb3b0318e38fab1a02dd2b.

VCP-04 | CENTRALIZATION RELATED RISKS

Category	Severity	Location	Status
Centralization	Centralization	ResilientOracle.sol (Update1): 321; oracles/common/CorrelatedTokenOracle.sol (Update1): 120	48h Timelock

Description

Note that any centralization risks present in the existing codebase before the PR's in scope of this audit were not considered. Only those added to the in-scope PRs are addressed. We recommend all users carefully review the centralization risks, much of which can be found in our previous audits, which can be found here: https://skynet.certik.com/projects/venus.

In the contract `CorrelatedTokenOracle`, the role `DEFAULT_ADMIN_ROLE` of the `AccessControlManager` can grant addresses the privilege to call the following functions:

- `setSnapshot()`
- `setGrowthRate()`
- `setSnapshotGap()`

Any compromise to the `DEFAULT_ADMIN_ROLE` or accounts granted this privilege may allow the hacker to take advantage of this authority and do the following:

- Set the snapshot exchange rate and timestamp to any value, allowing them to effectively bypass the cap behavior.
- Set the growth rate to any value, allowing them to effectively bypass the cap behavior.
- Set the snapshot gap to any value, allowing them to effectively bypass the cap behavior.

In the contract `ResilientOracle`, the role `DEFAULT_ADMIN_ROLE` of the `AccessControlManager` can grant addresses the privilege to call the following functions:

- `setTokenConfig()`

This function was adapted to also allow caching of prices to be enabled or disabled. Any compromise to the `DEFAULT_ADMIN_ROLE` or accounts granted this privilege may allow the hacker to take advantage of this authority and make calls less efficient by disabling cacheing or cacheing prices of assets whose price can change in the same transaction to them exploit the protocol due to the price discrepancy.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend

centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[Venus, 04/29/2025]: Only the Normal Timelock on each network has the `DEFAULT_ADMIN_ROLE` on the `AccessControlManager` contract. See Governance for the list of Normal Timelock and `AccessControlManager` contracts. So, only via a Normal VIP (involving the Venus Community) will be possible to:

- grant permissions to execute the following functions on the CorrelatedTokenOracle: `setSnapshot`, `setGrowthRate`, `setSnapshotGap`
- grant permissions to execute the function `ResilientOracle.setTokenConfig`

The Normal, Fast-Track, and Critical Timelocks on each network will have permissions to invoke the mentioned functions.

CTO-08 | ISSUES WITH EXCHANGE RATE MANIPULATIONS

Category	Severity	Location	Status
Logical Issue	Medium	oracles/common/CorrelatedTokenOracle.sol (Base): 98	Resolved

Description

The function `updateSnapshot()` can be called by anyone. An attacker can use this function to update the snapshot while the exchange rate is manipulated, which can cause the oracle to return inaccurate prices.

In addition, if an attacker can manipulate the exchange rate down, it is not bounded. An attacker could manipulate the exchange rate down prior to `getPrice()` being called so that the price is significantly lower than the actual price. This could allow positions to be liquidated, when they should not be.

Scenario

Assume that an attacker can manipulate the exchange rate down and that the current exchange rate is 2e18. Furthermore assume that the `SNAPSHOT_INTERVAL` has just passed.

- The attacker manipulates the exchange rate down, so that it is 1e18.
- In the same transaction the attacker calls `updateSnapshot()`.
    - The fetched manipulated exchange rate of 1e18 is less than the `maxAllowedExchangeRate` so that `snapshotExchangeRate` is updated to be `1e18`, significantly less than the true exchange rate.
- The attacker reverses the exchange rate manipulation so that the exchange rate is back to 2e18.
- All subsequent calls to `getPrice()` will then use the `snapshotExchangeRate = 1e18` to calculate the `maxAllowedExchangeRate`, so that the returned `maxAllowedExchangeRate` is significantly less than the actual exchange rate. Thus it will use the `maxAllowedExchangeRate` to calculate the price, which will cause the assets price to be significantly less than the actual price.

In this way an attacker can manipulate the exchange rate during a snapshot update in order to cause the oracle to return prices that are lower than the actual price.

Assume that an attacker can manipulate the exchange rate up and the current exchange rate is 2e18. Furthermore assume that the `SNAPSHOT_INTERVAL` has just passed.

- The attacker manipulates the exchange rate up to be 3e18.
- In the same transaction the attacker calls `updateSnapshot()`.
    - The exchange rate is higher than the `maxAllowedExchangeRate` so that the `snapshotExchangeRate` will be set to be the `maxAllowedExchangeRate`, even if the true exchange rate is not the maximum.

- The attacker reverses the exchange rate manipulation so that the exchange rate is back to 2e18.

The attacker can do this at every snapshot to ensure that the stored `snapshotExchangeRate` is always the maximum amount possible. This allows for the actual exchange rate and the `snapshotExchangeRate` to differ more each snapshot. If the difference becomes significant, then the attacker can manipulate the exchange rate prior to `getPrice()` being called to return a price that is significantly higher that the price of the asset.

In this way an attacker can continuously manipulate the exchange rate during snapshot updates to cause the `snapshotExchangeRate` to be significantly higher than the actual exchange rate, allowing manipulations of the exchange rate to cause the oracle to return prices significantly higher than the actual price.

Recommendation

We recommend ensuring attackers cannot use `updateSnapshot()` to store a manipulated exchange rate as the `snapshotExchangeRate`. In addition, we recommend ensuring that no token whose exchange rate can be manipulated down is supported or ensuring that the exchange rate is also bounded below with a minimum exchange rate.

Alleviation

`[Venus, 03/10/2025]`: "Regarding the scenario where the attacker manipulates the exchange rate down, some ERC4626 tokens support decreases of the exchange rate, performed by priviledged accounts. For example, burning underlying tokens, or minting yield tokens without providing any underlying assets. They could have these mechanisms to adjust the exchange rate after an incident, for example.

If we set a lower bound for the exchange rate, and the admins of the token decrease the exchange rate, our oracle would consider an inflated price, and that could have negative side effects. For example, suppliers would have a greater borrowing power, and they could generate a bad debt in the protocol.

As a reference, other protocols, like AAVE (https://github.com/bgd-labs/aave-capo/blob/main/src/contracts/PriceCapAdapterBase.sol) don't include lower bounds in their capped oracles.

So, we think it's ok to not add a lower bound to the exchange rate, assuming only privilege accounts will be able to decrease that.

Regarding the scenario where the attacker manipulates the exchange rate up, there are two options:

1. The attacker reverses the exchange rate manipulation in the same transaction. For example, with privileged rights on the token. We suppose that if the attacker could do that, they would take advantage of it in a different way, for example, redeeming the tokens considering a greater exchange rate. But, focused on our oracle, the attacker would be able to update the snapshot to the maximum allowed each time, no more. So, only after several updates there could be a significant allowed difference with the real exchange rate, that could make the attack profitable. We'll deploy monitoring tasks to be notified when the exchange rate returned by the contract is lower than the current snapshot. This will be useful for the first scenario of this finding.

2. The attacker doesn't reverse the manipulation in the same transaction. For example, with a donation attack. In that case, the exchange rate returning by the contract after the manipulation is the "real" one. Our oracle would cap it

> anyway, and if the snapshot has to be updated in that period of time, it will be increased to a limited new value"

`[CertiK, 03/12/2025]`: Thank you for the comments. Considering the comments and assumptions we have reduced the severity of this finding to medium. However, we have the following comments:

In regards to lower bound on the exchange rate, it may be desired to revert in such a scenario. In particular, the privileged roles may become compromised in which case they can target the Venus protocol in order to steal more funds. One potential mitigation would be to set a lower bound that would always be upheld during normal operations, then if there is an abnormal decrease in the exchange rate it would cause a revert. Provided there is advance notice of such abnormal decreases parameters can be adjusted to prevent a denial of service, however, if there is a potential for a denial of service in such cases which may cause further issues in the protocol. We recommend considering such a mechanism to reduce the trust assumption on these privileged roles, however, we recognize there may be unforeseen issues in such a mechanism.

In regard to the exchange rate manipulation we agree with case 2. For case 1, there is the following scenario that we have a concern:

- Assume that the snapshot is just updated to the current exchange rate of 2e18. Assume for simplicity the cap is 2.1e18.
- By some means (potentially a privileged action) the exchange rate is decreased significantly to 1.5e18 before the snapshot interval has passed.
- An attacker can then manipulate the exchange rate from 1.5e18 up to 2.1e18 causing a significant price difference that can allow a profitable attack.

While there may be other avenues that the attacker would attempt to make a profitable attack if they can do such a manipulation, there is still a possibility that they can attack the Venus protocol.

`[Venus, 04/16/2025]`: "We have added privilege functions to set the snapshot data (value and timestamp) manually, via Governance. That way, for those assets with a risk of exchange rate manipulation, the snapshots will be performed manually (setting SNAPSHOT_INTERVAL to max uint256, for example, and calling the new `setSnapshot` function periodically)"

`[CertiK, 04/22/2025]`: The client made the changes mentioned above in commit b5d7106c5e5e3844d4219dcb163920d4dcda9800. We assume that assets will be throughly vetted and those at risk of an exchange rate manipulation will be handled as mentioned above, so we mark this finding as *Resolved*.

VPB-01 DIFFERING BEHAVIOR IF ORACLES PRICE DIFFERS DURING TRANSACTION

Category	Severity	Location	Status
Logical Issue	Medium	ResilientOracle.sol (Base): 334~336, 358~361; oracles/common/CorrelatedTokenOracle.sol (Base): 99~101, 120~123	Resolved

Description

In the `ResilientOracle`, if `updatePrice()` or `updateAssetPrice()` is called, it will fetch the price and then cache it. Any subsequent calls to get the price in the same transaction will then return the cached price. If an oracle that may return a different price during the same transaction is supported, it will differ from the current behavior of the protocol and may cause unforeseen consequences.

Similarly, in the `CorrelatedTokenOracle`, if `updateSnapshot()` is called, it will fetch the current exchange rate and cache it provided the `SNAPSHOT_INTERVAL` has passed since the last update and it is nonzero. Any subsequent calls to get the price in the same transaction will then use the cached exchange rate. If an oracle that may return a different exchange rate during the same transaction is supported, it will differ from the current behavior of the protocol and may cause unforeseen consequences. Note that `updateSnapshot()` is called within `updatePrice()` and `updateAssetPrice()` as well.

For example, assume that the Pyth oracle is the only oracle used for an asset and that `updateAssetPrice()` is called and caches the current price returned by the Pyth oracle. In the same transaction a user calls `updatePriceFeeds()` to have the Pyth oracle prices updated so that the price of the asset is updated and differs from the cached value. Any further actions in the same transaction will use the cached price which differs from the current price that would be returned by the Pyth oracle.

Also if there are actions performed during the transaction that affect the exchange rate of an asset, then this can cause a discrepancy between the cached exchange rate and the true current exchange rate.

Recommendation

We recommend careful consideration of all cases where the cached price may differ from the current price to ensure that using the cached price over the current price is desired.

Alleviation

`[Venus, 03/10/2025]`:

1. We don't support the Pyth Oracle anymore. We don't have any oracle configuration using it, and we won't do it in the near future. So, the provided example is not applicable

2. We think caching the exchange rate shouldn't increase any risk. In fact, it acts like a security measure against exchange rate manipulations (with a donation attack, for example)

We are already using "cached" prices, and "cached" exchange rates, when we read the prices (or exchange rates) from the feeds provided by the Oracle companies (Chainlink and RedStone, for example). These companies don't update their feeds during a transaction. And it seems acceptable.

Do you see any specific security risk with our caching policy"

`[CertiK, 03/12/2025]`: Thank you for the comments. Provided the oracles use cached prices that don't update their feeds in a way that it can be included in a malicious transaction we agree there is no concern in using cached prices from oracles.

In regards to exchange rates the following scenario is of concern:

1. Assume that an ERC4626 style correlated token is supported and they do not use virtual shares and do not lock an initial amount of tokens. Furthermore assume for simplicity that the current exchange rate is 2e18 and the initial exchange rate is 1e18.
2. Assume that all but one single malicious user withdraw from the ERC4626 vault, but the token is still supported as a lending market.
3. They then execute the following in a single transaction.
    - The malicious user then updates the snapshot in the oracle so that it caches the current exchange rate of 2e18.
    - The malicious user withdraws all their assets so that the vault is empty.
    - The malicious user deposits the assets agains, however, as the vault was empty they will be minted shares according to the initial exchange rate of 1e18. Doubling the amount of the ERC4626 token they hold.
    - The user then supplies all their ERC4626 token in the lending protocol, however, it will use the cached exchange rate of 2e18 so that it will assume the user deposited 2 times the actual amount that was deposited.
    - The user then borrows as much as possible against their deposited amount. Provided they can borrow more than 50% of the value, their position can become undercollateralized stealing value from the protocol.

Note that the typical mitigation for the hundred finance style exploit of locking an initial supplied amount of tokens will ensure that the ERC4626 vault always has a non-malicious depositor and prevents this scenario. However, we feel it is important to consider this scenario independently in case the protocol is adapted to no longer lock an initial amount of tokens when a new market is supported in the future.

This demonstrates one way that a cached exchange rate may not be the exchange rate that should be used. It also demonstrates a way that the exchange rate can be manipulated down by a non privileged entity.

Note that if a privileged entity can manipulate the exchange rate down, then they can do a similar attack to exploit the difference in the cached exchange rate. While if the privileged role is compromised they may do other actions to gain value,

the attacker could still target the Venus protocol to steal more funds by draining all funds from the pool that supports the market.

`[Venus, 04/16/2025]`: "We have removed the cache of exchange rates. It will be always read from the associated contract.

Moreover, we have removed the interaction with TWAP oracles. That was dead code.

Finally, we have added a flag to the struct defining the oracle configuration per asset, making the cache optional. That way, it will be doable to disable the cache if the USD value of the underlying token can change during the same transaction.

The main policy will be to disable the cache for the assets whose USD price depends on an exchange rate, recovered on-chain, that can change in the same transaction."

`[CertiK, 04/22/2025]`: The client made the changes mentioned above in commit 65c06ef4a668abe54cd09b7d08fdfe2b102226ac. We assume that assets will be thoroughly vetted and those whose price or exchange rate may change in the same transaction will have their caching disabled, so we mark this finding *Resolved*.

CTO-02 | MISSING INPUT VALIDATION

Category	Severity	Location	Status
Logical Issue	Minor	oracles/common/CorrelatedTokenOracle.sol (Base): 63~64, 74	Resolved

Description

In the contract `CorrelatedTokenOracle`:

- In the constructor, the `annualGrowthRate` is used in checks, when the value used in the contract is `GROWTH_RATE_PER_SECOND` which is derived from it. This allows for the case where the `GROWTH_RATE_PER_SECOND` is 0 while the `snapshotInterval` is nonzero, which can cause unexpected behavior.

Recommendation

We recommend adding the input validations mentioned above.

Alleviation

`[CertiK, 03/10/2025]`: The client made the recommended changes resolving the finding in commit 0f552cc4499cfd209363ded0d5fdafbc756e6a9d.

CTO-03 | `isCapped()` MAY RETURN TRUE WHEN IT IS NOT CAPPED

Category	Severity	Location	Status
Logical Issue	Minor	oracles/common/CorrelatedTokenOracle.sol (Base): 84~93, 127~129	Resolved

Description

If the `SNAPSHOT_INTERVAL = 0`, then when `getPrice()` is called it skips checks for the maximum exchange rate so that there is no cap. However, if during construction the input `initialSnapshotExchangeRate` is nonzero, then `isCapped()` may return `true`, when there is no cap.

This is because it will call `_getMaxAllowedExchangeRate()`, which will return `snapshotExchangeRate`. `snapshotExchangeRate` is only set in the constructor as `updateSnapshot()` does not update its value if `SNAPSHOT_INTERVAL = 0`. If the current exchange rate is larger than the `initialSnapshotExchangeRate`, then `isCapped()` will return true. For example this will likely happen if the `initialSnapshotExchangeRate` is set to be the exchange rate at the time of deployment.

Recommendation

We recommend ensuring that `isCapped()` returns `false` if `SNAPSHOT_INTERVAL = 0`.

Alleviation

`[CertiK, 03/10/2025]`: The client made the recommended changes resolving the finding in commit 111b75896ca5903bb09fd19bce5f1975d55448b6.

CTO-04 | ISSUE IF `maxAllowedExchangeRate = 0`

Category	Severity	Location	Status
Logical Issue	Minor	oracles/common/CorrelatedTokenOracle.sol (Base): 133	Resolved

Description

In the function `getPrice()`, if `maxAllowedExchangeRate = 0`, then it will return the `exchangeRate` as opposed to the `maxAllowedExchangeRate`. It is the return value of `_getMaxAllowedExchangeRate()`, which can only be zero if the `snapshotExchangeRate = 0`.

`snapshotExchangeRate` can only be 0 if the `initialSnapshotExchangeRate` is input as 0 or if during an update `_getUnderlyingAmount()` returns zero. The first case can be prevented by adding input validation or ensuring a value of 0 is not input at construction if it should be capped. For the second case, if a user can manipulate the exchange rate to be zero during the snapshot update, then they can manipulate the exchange rate to be any value and avoid the cap until the next snapshot.

Recommendation

We recommend ensuring that the `initialSnapshotExchangeRate` is nonzero provided the exchange rate is capped. In addition, we recommend explicitly handling the case if `snapshotExchangeRate` is set to 0 during a snapshot update.

Alleviation

`[CertiK, 03/10/2025]`: The client made the recommended changes resolving the finding in commit d0b5c9a9108292d60906bd94fd5ff01896e1f2b7.

VCP-01 | DISCUSSION ON `ACCESS_CONTROL_MANAGER`

Category	Severity	Location	Status
Inconsistency	Minor	oracles/common/CorrelatedTokenOracle.sol (Update1): 26~27	Resolved

Description

Currently, the `ACCESS_CONTROL_MANAGER` is only set in the `constructor()` and has the naming convention of an immutable variable. In addition, the contract is not designed to be upgradeable, so it does not inherit `AccessControlledV8` and instead includes the `_checkAccessAllowed()` function explicitly.

Recommendation

We recommend adding the `immutable` specifier for `ACCESS_CONTROL_MANAGER` field.

Alleviation

[Venus, 04/30/2025]: The team heeded the advice and resolved the issue by in 0395e8185fd94d3702194b993cde8a9f1864eed4

CTO-05 | EVENT NOT INDEXED

Category	Severity	Location	Status
Design Issue	Informational	oracles/common/CorrelatedTokenOracle.sol (Base): 44	Resolved

Description

If an event is not indexed in a smart contract, it means that the event's parameters are not tagged with the `indexed` keyword. This has implications for how the event data can be searched and filtered when looking through blockchain logs.

Without indexing, the event will still emit the data as part of the transaction log, but users won't be able to query for these events using the parameters. They'll have to retrieve the entire set of logs and manually sift through them to find events with the specific data. This can be less efficient and more time-consuming, especially on a blockchain with a high volume of transactions and events.

Recommendation

We recommend indexing the most relevant parameters in the event to be defined.

Alleviation

`[CertiK, 03/10/2025]`: The client made the recommended changes resolving the finding in commit 21f6448c2ccb5693d72d19ea8f36f999af6bc996.

CTO-06 INITIAL VALUES SHOULD BE VERIFIED BEFORE DEPLOYMENT

Category	Severity	Location	Status
Logical Issue	Informational	oracles/common/CorrelatedTokenOracle.sol (Base): 76~77	Acknowledged

Description

The input `initialSnapshotExchangeRate` and `initialSnapshotTimestamp` should be verified prior to deployment to be accurate as full input validation cannot be added in all cases. This is because a manipulation of the exchange rate could cause a denial of service if the current exchange rate is used and not all contracts will have the ability to verify past exchange rates.

Recommendation

We recommend ensuring that the input `initialSnapshotExchangeRate` corresponds to the exchange rate at the `initialSnapshotTimestamp` .

Alleviation

`[Venus, 03/07/2025]` : "Issue acknowledged. I won't make any changes for the current version.

During the VIP (Venus Improvement Proposals) and simulation stage, the values will be verified for a given timestamp and block number"

CTO-07 | CONTRACT DOESN'T INHERIT INTERFACE

Category	Severity	Location	Status
Logical Issue	Informational	oracles/common/CorrelatedTokenOracle.sol (Base): 13	Resolved

Description

The contract `CorrelatedTokenOracle` implements the `OracleInterface` and the `ICappedOracle` interface, however, it does not inherit them. This can cause issues if changes are made that may not make it compatible with the interfaces it is designed to implement.

Recommendation

We recommend inheriting the interfaces that a contract implements to ensure their compatability.

Alleviation

`[CertiK, 03/06/2025]`: The client made the recommended changes resolving the finding in commits

- 4f6d97527252eb3209f1dbc31edcf2fe19f5ece8;
- 6f77ab693a522fcc714f95f11e5cffe8d8836ff0.

VCP-02 | LOCAL VARIABLE SHADOWING

Category	Severity	Location	Status
Coding Style	Informational	oracles/AnkrBNBOracle.sol (Update1): 22, 26; oracles/AsBNBOracle.sol (Update1): 22, 26; oracles/BNBxOracle.sol (Update1): 27, 31; oracles/BinanceOracle.sol (Update1): 62; oracles/ERC4626Oracle.sol (Update1): 21, 25; oracles/EtherfiAccountantOracle.sol (Update1): 25, 29; oracles/OneJumpOracle.sol (Update1): 25, 29; oracles/SFraxOracle.sol (Update1): 20, 24; oracles/SFrxETHOracle.sol (Update1): 58; oracles/SlisBNBOracle.sol (Update1): 27, 31; oracles/StkBNBOracle.sol (Update1): 30, 34; oracles/WBETHOracle.sol (Update1): 19, 23; oracles/WeETHAccountantOracle.sol (Update1): 24, 28; oracles/WeETHOracle.sol (Update1): 25, 29; oracles/WstETHOracleV2.sol (Update1): 20, 24; oracles/ZkETHOracle.sol (Update1): 19, 23; oracles/common/CorrelatedTokenOracle.sol (Update1): 32, 41	Resolved

Description

A local variable is shadowing another component defined elsewhere. This means that when the contract accesses the variable by its name, it will use the one defined locally, not the one defined in the other place. While this is currently used correctly, updates to the codebase may not handle this correctly and cause issues in the future.

```
20          uint256 snapshotInterval,
```

- Local variable `snapshotInterval` in `SFraxOracle.constructor` shadows the variable `snapshotInterval` in `CorrelatedTokenOracle`.

```
24          uint256 snapshotGap
```

- Local variable `snapshotGap` in `SFraxOracle.constructor` shadows the variable `snapshotGap` in `CorrelatedTokenOracle`.

```
26          uint256 snapshotGap
```

- Local variable `snapshotGap` in `AsBNBOracle.constructor` shadows the variable `snapshotGap` in `CorrelatedTokenOracle`.

```
22          uint256 snapshotInterval,
```

- Local variable `snapshotInterval` in `AsBNBOracle.constructor` shadows the variable `snapshotInterval` in `CorrelatedTokenOracle`.

```
27          uint256 snapshotInterval,
```

- Local variable `snapshotInterval` in `BNBxOracle.constructor` shadows the variable `snapshotInterval` in `CorrelatedTokenOracle`.

```
31          uint256 snapshotGap
```

- Local variable `snapshotGap` in `BNBxOracle.constructor` shadows the variable `snapshotGap` in `CorrelatedTokenOracle`.

```
23          uint256 snapshotGap
```

- Local variable `snapshotGap` in `WBETHOracle.constructor` shadows the variable `snapshotGap` in `CorrelatedTokenOracle`.

```
19          uint256 snapshotInterval,
```

- Local variable `snapshotInterval` in `WBETHOracle.constructor` shadows the variable `snapshotInterval` in `CorrelatedTokenOracle`.

```
28          uint256 snapshotGap
```

- Local variable `snapshotGap` in `WeETHAccountantOracle.constructor` shadows the variable `snapshotGap` in `CorrelatedTokenOracle`.

```
24          uint256 snapshotInterval,
```

- Local variable `snapshotInterval` in `WeETHAccountantOracle.constructor` shadows the variable `snapshotInterval` in `CorrelatedTokenOracle`.

```
21          uint256 snapshotInterval,
```

- Local variable `snapshotInterval` in `ERC4626Oracle.constructor` shadows the variable `snapshotInterval` in `CorrelatedTokenOracle`.

```
25          uint256 snapshotGap
```

- Local variable `snapshotGap` in `ERC4626Oracle.constructor` shadows the variable `snapshotGap` in `CorrelatedTokenOracle`.

```
29          uint256 snapshotGap
```

- Local variable `snapshotGap` in `OneJumpOracle.constructor` shadows the variable `snapshotGap` in `CorrelatedTokenOracle`.

```
25          uint256 snapshotInterval,
```

- Local variable `snapshotInterval` in `OneJumpOracle.constructor` shadows the variable `snapshotInterval` in `CorrelatedTokenOracle`.

```
29          uint256 snapshotGap
```

- Local variable `snapshotGap` in `EtherfiAccountantOracle.constructor` shadows the variable `snapshotGap` in `CorrelatedTokenOracle`.

```
25          uint256 snapshotInterval,
```

- Local variable `snapshotInterval` in `EtherfiAccountantOracle.constructor` shadows the variable `snapshotInterval` in `CorrelatedTokenOracle`.

```
58      function initialize(address _accessControlManager, uint256
 _maxAllowedPriceDifference) external initializer {
```

- Local variable `_accessControlManager` in `SFrxETHOracle.initialize` shadows the variable `_accessControlManager` in `AccessControlledV8`.

```
30          uint256 snapshotInterval,
```

- Local variable `snapshotInterval` in `StkBNBOracle.constructor` shadows the variable `snapshotInterval` in `CorrelatedTokenOracle`.

```
34          uint256 snapshotGap
```

- Local variable `snapshotGap` in `StkBNBOracle.constructor` shadows the variable `snapshotGap` in `CorrelatedTokenOracle`.

```
62      function initialize(address _sidRegistryAddress, address
 _accessControlManager) external initializer {
```

- Local variable `_accessControlManager` in `BinanceOracle.initialize` shadows the variable `_accessControlManager` in `AccessControlledV8`.

```
26          uint256 snapshotGap
```

- Local variable `snapshotGap` in `AnkrBNBOracle.constructor` shadows the variable `snapshotGap` in `CorrelatedTokenOracle`.

```
22          uint256 snapshotInterval,
```

- Local variable `snapshotInterval` in `AnkrBNBOracle.constructor` shadows the variable `snapshotInterval` in `CorrelatedTokenOracle`.

```
31          uint256 snapshotGap
```

- Local variable `snapshotGap` in `SlisBNBOracle.constructor` shadows the variable `snapshotGap` in `CorrelatedTokenOracle`.

```
27          uint256 snapshotInterval,
```

- Local variable `snapshotInterval` in `SlisBNBOracle.constructor` shadows the variable `snapshotInterval` in `CorrelatedTokenOracle`.

```
20          uint256 snapshotInterval,
```

- Local variable `snapshotInterval` in `WstETHOracleV2.constructor` shadows the variable `snapshotInterval` in `CorrelatedTokenOracle`.

```
24          uint256 snapshotGap
```

- Local variable `snapshotGap` in `WstETHOracleV2.constructor` shadows the variable `snapshotGap` in

`CorrelatedTokenOracle`.

```
25          uint256 snapshotInterval,
```

- Local variable `snapshotInterval` in `WeETHOracle.constructor` shadows the variable `snapshotInterval` in `CorrelatedTokenOracle`.

```
29          uint256 snapshotGap
```

- Local variable `snapshotGap` in `WeETHOracle.constructor` shadows the variable `snapshotGap` in `CorrelatedTokenOracle`.

```
23          uint256 snapshotGap
```

- Local variable `snapshotGap` in `ZkETHOracle.constructor` shadows the variable `snapshotGap` in `CorrelatedTokenOracle`.

```
19          uint256 snapshotInterval,
```

- Local variable `snapshotInterval` in `ZkETHOracle.constructor` shadows the variable `snapshotInterval` in `CorrelatedTokenOracle`.

Recommendation

We recommend renaming the local variable that shadows another definition to prevent potential issues and maintain the expected behavior of the smart contract.

Alleviation

[Venus, 04/30/2025]: The team heeded the advice and resolved the issue in commit 105d3c50af77caa7a731167ad2ca91bb927dcc20.

VCP-03 | SPECIFIC IMPORTS ARE NOT USED

Category	Severity	Location	Status
Coding Style	Informational	ResilientOracle.sol (Update1): 5~11; oracles/common/CorrelatedTokenOracle.sol (Update1): 9	Resolved

Description

Specific imports are used consistently throughout the codebase, however, it is not used in the imports cited above.

Recommendation

We recommend using specific imports consistently throughout the codebase.

Alleviation

[Venus, 04/30/2025]: The team heeded the advice and resolved the issue in commit 995b4381dd516901c6d99ef90e42197cd67255b4, 0a42dacf50222bb8d2f73974db001c2b05345caa.

VCP-05 | MISLEADING NAME

Category	Severity	Location	Status
Inconsistency	Informational	oracles/common/CorrelatedTokenOracle.sol (Update1): 34~35, 188~190	Resolved

Description

The `snapshotExchangeRate` is not longer the exchange rate at the `snapshotTimestamp` or the capped exchange rate. It is either the max allowed exchange rate at that time plus the gap or the exchange rate plus the gap. With the name `snapshotExchangeRate` it may be expected to be either the exchange rate or capped exchange rate at the snapshot, which it is not.

Recommendation

We recommend renaming the variable to something like `snapshotMaxExchangeRate` to clarify that it is a snapshotted value used to determine the max exchange rate.

Alleviation

[Venus, 04/30/2025]: The team heeded the advice and resolved the issue in commit 666a2310e078fb1e5e2c910218484668a0883c9b.

VPB-02 | `CACHE_SLOT` DOES NOT FOLLOW ERC-7201

Category	Severity	Location	Status
Logical Issue	Informational	ResilientOracle.sol (Base): 87; lib/Transient.sol (Base): 4; oracles/common/CorrelatedTokenOracle.sol (Base): 14~15	Resolved

Description

The `CACHE_SLOT` is determined by the hash of an encoded string (namespace). However, it does not follow the namespaced storage layout ERC-7201. As the method used is not standardized, it may not be compatible with tools where these design patterns are used.

Recommendation

We recommend using `ERC-7201` for the namespace storage layout to help ensure its compatibility.

Alleviation

`[CertiK, 03/10/2025]`: The client made the recommended changes in commits

- 73e00238e0aa297fc419b7bff81e61ea348e8abf;
- caf262d95d52513a216e424e7784b8de4762f644.

VPB-03 | MISSING/INCOMPLETE NATSPEC

Category	Severity	Location	Status
Inconsistency	Informational	ResilientOracle.sol (Base): 87, 230, 354; oracles/PendleOracle.sol (Base): 108; oracles/StkBNBOracle.sol (Base): 58; oracles/common/CorrelatedTokenOracle.sol (Base): 14, 64, 95~97, 147; ResilientOracle.sol (Update1): 299~308, 493~495; oracles/common/CorrelatedTokenOracle.sol (Update1): 119, 133, 151, 260	Resolved

Description

The following functions NatSpec comments are incomplete or missing.

CorrelatedTokenOracle

- The NatSpec comments for the function `updateSnapshot()` do not include the event that is emitted.
- The NatSpec comments for the function `calculatePrice()` do not include the error that can be emitted.
- The NatSpec comments for the constructor do not include the error that can be emitted.
- The variable `CACHE_SLOT` has a comment, but does not use "@notice".

PendleOracle

- The NatSpec comments for the constructor do not include the error that can be emitted.

StkBNBOracle

- The NatSpec comments for the function `_getUnderlyingAmount()` do not include the error that can be emitted.

ResilientOracle

- The NatSpec comments for the function `enableOracle()` do not include the event that is emitted.
- The function `_getPrice()` does not have NatSpec comments.
- The variable `CACHE_SLOT` does not have a NatSpec comment.

In addition, the following missing or incomplete NatSpec comments where found during the second audit of the contracts:

ResilientOracle

- The function `_isCacheEnabled()` does not have NatSpec comments.
- The NatSpec comments for the function `setTokenConfig()` does not include the event `CachedEnabled`.

CorrelatedTokenOracle

- The NatSpec comments for the function `setSnapshot()` do not include the event or errors that can be emitted.
- The NatSpec comments for the function `setGrowthRate()` do not include the event or errors that can be emitted.
- The NatSpec comments for the function `setSnapshotGap()` do not include the event or errors that can be emitted.
- The NatSpec comments for the function `_checkAccessAllowed()` do not include the event or errors that can be emitted.

Recommendation

We recommend completing or adding NatSpec comments for the functions listed above.

Alleviation

`[CertiK, 03/06/2025]`: The client made the recommended changes resolving the finding in commit

- c35811e3702ac95928fcf6afe20d83b51c24ed9a.
- c0e1ac777f82aca0e8e1b48bb664735abab63504.

VPB-04 | TYPOS AND INCONSISTENCIES

Category	Severity	Location	Status
Inconsistency	Informational	ResilientOracle.sol (Base): 234, 244, 329~331; oracles/AnkrBNBOracle.sol (Base): 17~18; oracles/BNBxOracle.sol (Base): 18~23; oracles/ERC4626Oracle.sol (Base): 16; oracles/EtherfiAccountantOracle.sol (Base): 16~20; oracles/OneJumpOracle.sol (Base): 16~20; oracles/PendleOracle.sol (Base): 35~56, 61~62; oracles/SFraxOracle.sol (Base): 14~15; oracles/SlisBNBOracle.sol (Base): 18~23; oracles/StkBNBOracle.sol (Base): 18~21, 25~26; oracles/WBETHOracle.sol (Base): 13~14; oracles/WeETHAccountantOracle.sol (Base): 14~19; oracles/WeETHOracle.sol (Base): 15~20; oracles/WstETHOracleV2.sol (Base): 14~15; oracles/common/CorrelatedTokenOracle.sol (Base): 18, 22, 26, 30, 34, 52~53, 159, 159, 163, 164; ResilientOracle.sol (Update1): 23, 41~44, 241, 251; oracles/AsBNBOracle.sol (Update1): 16; oracles/common/CorrelatedTokenOracle.sol (Update1): 114~118	Resolved

Description

ResilientOracle

- The comments above `updatePrice()`, `updateAssetPrice()`, and `_updateAssetPrice()` do not reflect that it updates the snapshot for the main oracle if it is a capped oracle.

CorrelatedTokenOracle

- The comments above the function `_getMaxAllowedExchangeRate()` state it returns a maximum price, when it returns a maximum exchange rate. In addition, the variable `maxPrice` is misleading as it stores the maximum exchange rate.
- Furthermore, throughout the codebase comments refer to the upgradability of contracts, which will no longer be the case as the proxy contracts are being removed.
- The function `calculatePrice()` is an internal function, but does not start with an underscore

---

In addition, the following typos and inconsistencies where found during the second audit of the contracts:

AsBNBOracle

- The comment above the `constructor()` for the contract `AsBNBOracle` refer to upgradability, which is no longer

the case as proxy contracts are being removed.

ResilientOracle

- Comments refer to the TWAP oracle, which was removed.

CorrelatedTokenOracle

- The comments above the function `setSnapshot()` do not include when this function is intended to be used.

Recommendation

We recommend fixing the typos and inconsistencies mentioned above.

Alleviation

`[CertiK, 03/10/2025]`: The client made the recommended changes resolving this finding in commits

- 4f6d97527252eb3209f1dbc31edcf2fe19f5ece8;
- e417976e5b4bb11ae0fd61dbdc8eec56e7d34de1.
- edb889bd094d11e3c828e0d0e4f752d1bf005f4b.

OPTIMIZATIONS

VENUS - CACHED PRICES AND CAPPED ORACLES

ID	Title	Category	Severity	Status
CTO-01	Check On Input Asset Can Be Performed Earlier	Gas Optimization	Optimization	Resolved

CTO-01 | CHECK ON INPUT ASSET CAN BE PERFORMED EARLIER

Category	Severity	Location	Status
Gas Optimization	Optimization	oracles/common/CorrelatedTokenOracle.sol (Base): 147	Resolved

Description

The function `calculatePrice()` checks that the input `asset` is the `CORRELATED_TOKEN`. However, it is always called with the input `asset` of the function `getPrice()`. As such, the input `asset` can be checked in the function `getPrice()` so that it will revert sooner and save users gas in case of a revert.

Recommendation

We recommend checking the input `asset` in the function `getPrice()` as opposed to the function `calculatePrice()`.

Alleviation

`[CertiK, 03/10/2025]`: The client made the recommended changes resolving the finding in commit 7b2dcab30c87cdc3b9d87cf86c1631db1053afb6.

FORMAL VERIFICATION

VENUS - CACHED PRICES AND CAPPED ORACLES

Formal guarantees about the behavior of smart contracts can be obtained by reasoning about properties relating to the entire contract (e.g. contract invariants) or to specific functions of the contract. Once such properties are proven to be valid, they guarantee that the contract behaves as specified by the property. As part of this audit, we applied formal verification to prove that important functions in the smart contracts adhere to their expected behaviors.

Considered Functions And Scope

In the following, we provide a description of the properties that have been used in this audit. They are grouped according to the type of contract they apply to.

Verification of Ownable2step Properties

We verified *partial* properties of the public interfaces of those token contracts that implement the Ownable2Step interface. This involves:

- function `owner` that returns the current owner,
- functions `renounceOwnership` that removes ownership,
- function `transferOwnership` that transfers the ownership to a new owner,
- function `pendingOwner` that return the pending owner, and
- function `acceptOwnership` by which a pending owner can accept the ownership.

The properties that were considered within the scope of this audit are as follows:

Property Name	Title
ownable2step-acceptownership-revert	`acceptOwnership` Prevents Invalid Inputs
ownable-transferownership-correct	Ownership is Transferred
ownable2step-acceptownership-succeed-normal	`acceptOwnership` Succeeds For Valid Inputs
ownable2step-pendingowner-succeed-normal	`pendingOwner` Always Succeeds
ownable-renounceownership-correct	Ownership is Removed
ownable-owner-succeed-normal	`owner` Always Succeeds

Verification Results

In the remainder of this section, we list all contracts where formal verification of at least one property was not successful. There are several reasons why this could happen:

- False: The property is violated by the project.
- Inconclusive: The proof engine cannot prove or disprove the property due to timeouts or exceptions.
- Inapplicable: The property does not apply to the project.

Detailed Results For Contract ResilientOracle (contracts/ResilientOracle.sol) In Commit 8c2e6d33585ab29c71a359c48054121b369ba1b3

Verification of Ownable2step Properties

Detailed Results for Function `acceptOwnership`

Property Name	Final Result	Remarks
ownable2step-acceptownership-revert	True	
ownable2step-acceptownership-succeed-normal	True	

Detailed Results for Function `transferOwnership`

Property Name	Final Result	Remarks
ownable-transferownership-correct	Inconclusive	

Detailed Results for Function `owner`

Property Name	Final Result	Remarks
ownable-owner-succeed-normal	True	

Detailed Results for Function `pendingOwner`

Property Name	Final Result	Remarks
ownable2step-pendingowner-succeed-normal	True	

Detailed Results for Function `renounceOwnership`

Property Name	Final Result	Remarks
ownable-renounceownership-correct	True	

Detailed Results For Contract BoundValidator (contracts/oracles/BoundValidator.sol) In Commit 8c2e6d33585ab29c71a359c48054121b369ba1b3

Verification of Ownable2step Properties

Detailed Results for Function `transferOwnership`

Property Name	Final Result	Remarks
ownable-transferownership-correct	False	

Detailed Results for Function `acceptOwnership`

Property Name	Final Result	Remarks
ownable2step-acceptownership-succeed-normal	True	
ownable2step-acceptownership-revert	True	

Detailed Results for Function `pendingOwner`

Property Name	Final Result	Remarks
ownable2step-pendingowner-succeed-normal	True	

Detailed Results for Function `renounceOwnership`

Property Name	Final Result	Remarks
ownable-renounceownership-correct	True	

Detailed Results for Function `owner`

Property Name	Final Result	Remarks
ownable-owner-succeed-normal	True	

Detailed Results For Contract ResilientOracle (contracts/ResilientOracle.sol) In Commit 17be0ef74817da3c9163f886d6e3d9aa6bd0c7ce

Verification of Ownable2step Properties

Detailed Results for Function `renounceOwnership`

Property Name	Final Result	Remarks
ownable-renounceownership-correct	True	

Detailed Results for Function `owner`

Property Name	Final Result	Remarks
ownable-owner-succeed-normal	True	

Detailed Results for Function `pendingOwner`

Property Name	Final Result	Remarks
ownable2step-pendingowner-succeed-normal	True	

Detailed Results for Function `acceptOwnership`

Property Name	Final Result	Remarks
ownable2step-acceptownership-succeed-normal	True	
ownable2step-acceptownership-revert	True	

Detailed Results for Function `transferOwnership`

Property Name	Final Result	Remarks
ownable-transferownership-correct	Inconclusive	

APPENDIX | VENUS - CACHED PRICES AND CAPPED ORACLES

Finding Categories

Categories	Description
Gas Optimization	Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.
Coding Style	Coding Style findings may not affect code behavior, but indicate areas where coding practices can be improved to make the code more understandable and maintainable.
Inconsistency	Inconsistency findings refer to different parts of code that are not consistent or code that does not behave according to its specification.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.
Design Issue	Design Issue findings indicate general issues at the design level beyond program logic that are not covered by other finding categories.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Details on Formal Verification

Some Solidity smart contracts from this project have been formally verified. Each such contract was compiled into a mathematical model that reflects all its possible behaviors with respect to the property. The model takes into account the semantics of the Solidity instructions found in the contract. All verification results that we report are based on that model.

The following assumptions and simplifications apply to our model:

- Certain low-level calls and inline assembly are not supported and may lead to a contract not being formally verified.
- We model the semantics of the Solidity source code and not the semantics of the EVM bytecode in a compiled contract.

Formalism for property specifications

All properties are expressed in a behavioral interface specification language that CertiK has developed for Solidity, which allows us to specify the behavior of each function in terms of the contract state and its parameters and return values, as well as contract properties that are maintained by every observable state transition. Observable state transitions occur when the contract's external interface is invoked and the invocation does not revert, and when the contract's Ether balance is changed by the EVM due to another contract's "self-destruct" invocation. The specification language has the usual Boolean connectives, as well as the operator `\old` (used to denote the state of a variable before a state transition), and several types of specification clause:

Apart from the Boolean connectives and the modal operators "always" (written `[]`) and "eventually" (written `<>`), we use the following predicates to reason about the validity of atomic propositions. They are evaluated on the contract's state whenever a discrete time step occurs:

- `requires [cond]` - the condition `cond`, which refers to a function's parameters, return values, and contract state variables, must hold when a function is invoked in order for it to exhibit a specified behavior.
- `ensures [cond]` - the condition `cond`, which refers to a function's parameters, return values, and both `\old` and current contract state variables, is guaranteed to hold when a function returns if the corresponding requires condition held when it was invoked.
- `invariant [cond]` - the condition `cond`, which refers only to contract state variables, is guaranteed to hold at every observable contract state.
- `constraint [cond]` - the condition `cond`, which refers to both `\old` and current contract state variables, is guaranteed to hold at every observable contract state except for the initial state after construction (because there is no previous state); constraints are used to restrict how contract state can change over time.

Description of the Analyzed Ownable2Step Properties

Properties related to function `acceptOwnership`

ownable2step-acceptownership-revert

Function `acceptOwnership` must always fail if called by a sender that is not the pending owner.

Specification:

```
reverts_when msg.sender != this.pendingOwner();
```

ownable2step-acceptownership-succeed-normal

Function `acceptOwnership` always succeeds if it is called by the pending owner.

Specification:

```
reverts_only_when msg.sender != this.pendingOwner();
```

Properties related to function `transferOwnership`

ownable-transferownership-correct

Invocations of `transferOwnership(newOwner)` must transfer the ownership to the `newOwner`.

Specification:

```
ensures this.owner() == newOwner;
```

Properties related to function `pendingOwner`

ownable2step-pendingowner-succeed-normal

Function `pendingOwner` must always succeed if it does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `renounceOwnership`

ownable-renounceownership-correct

Invocations of `renounceOwnership()` must set ownership to address(0).

Specification:

```
ensures this.owner() == address(0);
```

Properties related to function `owner`

ownable-owner-succeed-normal

Function `owner` must always succeed if it does not run out of gas.

Specification:

```
reverts_only_when false;
```

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

Elevating Your Entire Web3 Journey

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.

Copyright © CertiK