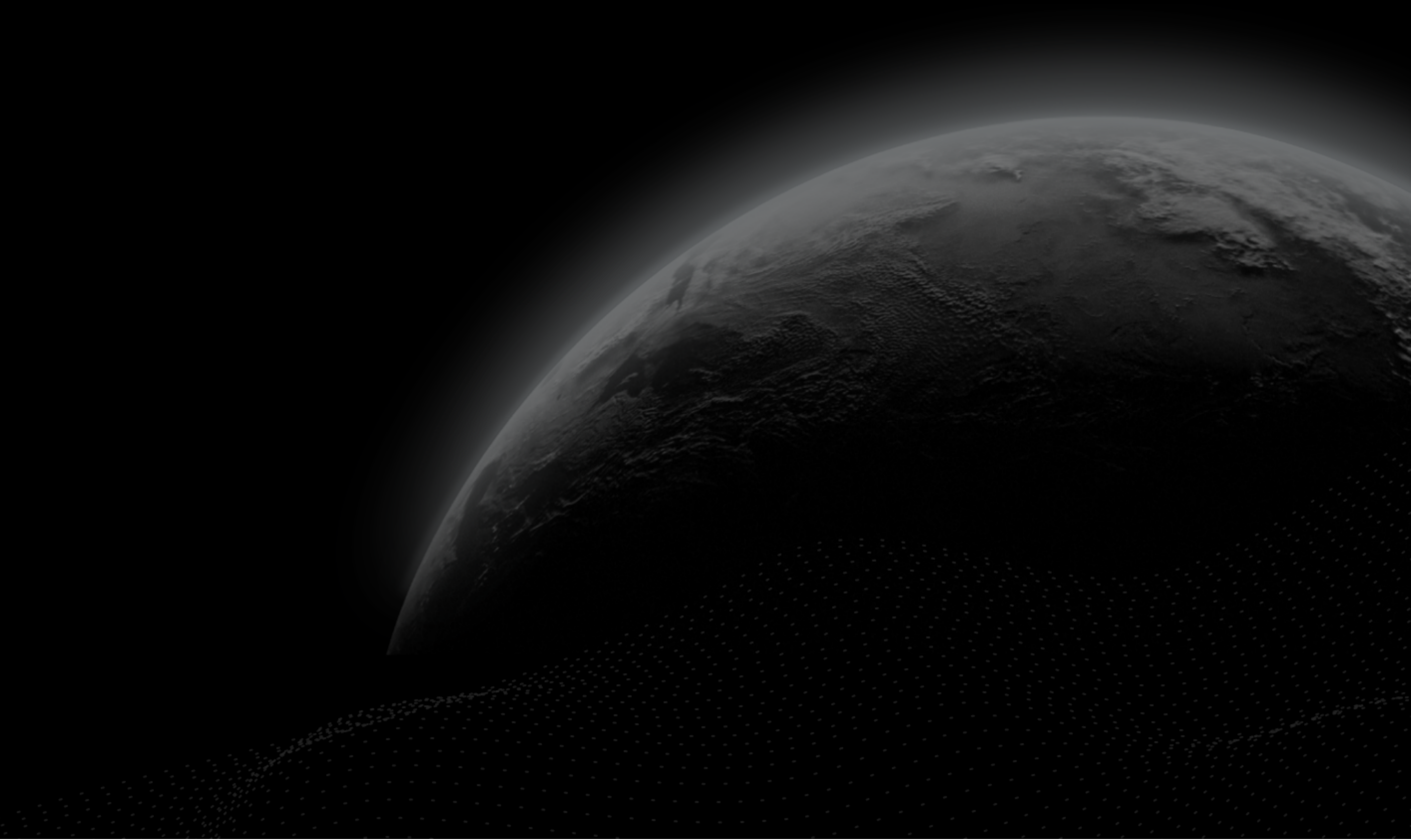
Security Assessment

Venus - Prime

CertiK Assessed on Nov 13th, 2023



CertiK Assessed on Nov 13th, 2023

Venus - Prime

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

Executive Summary

TYPES
DeFi

ECOSYSTEM
Binance Smart Chain (BSC)

METHODS
Manual Review, Static Analysis

LANGUAGE
Solidity

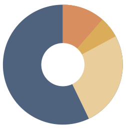
TIMELINE
Delivered on 11/13/2023

KEY COMPONENTS
N/A

CODEBASE
venus-protocol
View All in Codebase Page

COMMITS
base: af7c9afd7ce153778df2b9029b40a8f2a2359eeb
update1: 405f9629dd8ad4d17d447034e586c42bd43a2d0d
update2: fc6a76e29c6b59a03a9ca8f5b4072aa2b9492fa7
View All in Codebase Page

Vulnerability Summary



35	28	2	1	4	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined

🟥 0	Critical			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.
🟧 4	Major	2 Resolved, 2 Mitigated		Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
🟨 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.
🟨 9	Minor	6 Resolved, 1 Partially Resolved, 2 Acknowledged		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.
🟦 20	Informational	18 Resolved, 2 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 - PRIME

CODEBASE | VENUS - PRIME

Repository

venus-protocol

Commit

base: af7c9afd7ce153778df2b9029b40a8f2a2359eeb

update1: 405f9629dd8ad4d17d447034e586c42bd43a2d0d

update2: fc6a76e29c6b59a03a9ca8f5b4072aa2b9492fa7

update3: e7f211a4283595dec9484afd842afac25f6b43dc

update4: e02832bb2716bc0a178d910f6698877bf1b191e1

AUDIT SCOPE | VENUS - PRIME

24 files audited ● 4 files with Acknowledged findings ● 4 files with Mitigated findings ● 1 file with Partially Resolved findings ● 5 files with Resolved findings ● 10 files without findings

ID	Repo	Commit	File	SHA256 Checksum
● PLP	VenusProtocol/venus-protocol	af7c9af	Tokens/Prime/PrimeLiquidityProvider.sol	75e1acf69cc8c4deaf1d827c8967545422ca831b8e93fe4ea4903f0e4916bfcb
● PPT	VenusProtocol/venus-protocol	af7c9af	Tokens/Prime/Prime.sol	b7416261423d998ae1568cadb1b2edb2562262cda93d42f82dbcfbe69382820a
● FMT	VenusProtocol/venus-protocol	af7c9af	Tokens/Prime/libs/FixedMath0x.sol	dd6d8d46b4e808b4bd257bbd586ac057d6235a7d025e318adfb517e553661fe4
● PPV	VenusProtocol/venus-protocol	5926aa3	Tokens/Prime/Prime.sol	fc59b13e92a553e6368333ae3071dbeb8ed26590a33b47c64f63182a5103f4e6
● SFD	VenusProtocol/venus-protocol	af7c9af	Comptroller/Diamond/facets/SetterFacet.sol	f67c53e785534dbeb306c07adae77c26bafe7e1fa1082ceb5aca966c83f66e1d
● PFD	VenusProtocol/venus-protocol	af7c9af	Comptroller/Diamond/facets/PolicyFacet.sol	aed603903d41b429b510fa53b5d2ca8fe547b52855c834683e068812cbfea392
● XVS	VenusProtocol/venus-protocol	af7c9af	XVSVault/XVSVault.sol	70dd8b710e3d77f2c718d9e7a84ef5cf0eace9cb463d323db9a5a5143040b48b
● SPT	VenusProtocol/venus-protocol	af7c9af	Tokens/Prime/libs/Scores.sol	c84d35e912033e18e39b2ca0f55a48a0fee627c8ea17c3ad9d5b4217b94cb83a
● XVX	VenusProtocol/venus-protocol	5926aa3	XVSVault/XVSVault.sol	edb36e846dc915454300632adb0b68a1424a5973217c34316f0e53d096360ba4
● PSP	VenusProtocol/venus-protocol	af7c9af	Tokens/Prime/PrimeStorage.sol	76e8290f442d945c44851809260cf0ea85e7a71466c6ae6d00526455d0c0af1c

ID	Repo	Commit	File	SHA256 Checksum
🟢 FMP	VenusProtocol/venus-protocol	af7c9af	Tokens/Prime/libs/FixedMath.sol	cb267caba51af925a756932b4f30c939276263fd3f96d0a2dfd4dcb065dcae8d
🟢 SFC	VenusProtocol/venus-protocol	5926aa3	Comptroller/Diamond/facets/SetterFacet.sol	0a3b1ef6198f9d168e7c9eb0fc45781e6090d4c2b74301aba631a90451305250
🟢 CSV	VenusProtocol/venus-protocol	5926aa3	Comptroller/ComptrollerStorage.sol	aab128fdc51c04ba170037782b01d0779fd408b5bbfda1667e6960bcc82fc4f8
🟢 PLT	VenusProtocol/venus-protocol	5926aa3	Tokens/Prime/PrimeLiquidityProvider.sol	c1d6f56febbd242e1f375e1ac96ad81f3a6b0c8cda5c05dae5ecda23b9ad1639
⚪ CSC	VenusProtocol/venus-protocol	af7c9af	Comptroller/ComptrollerStorage.sol	defdef746d70266048ca6fc0cebcc36ba8a93e7bd505ebcf3ab5a12b00cf9ecd
⚪ XVV	VenusProtocol/venus-protocol	af7c9af	XVSVault/XVSVaultStorage.sol	b63e57a5f65136f8707e1a8c7830d6227b30004f1e59fd30d226dc74568980ea
⚪ IPP	VenusProtocol/venus-protocol	af7c9af	Tokens/Prime/IPrime.sol	47f5c7f0806931a5ba77efa6694e0ae25dc85cb1a78108f3146e8f7b6d426642
⚪ PFC	VenusProtocol/venus-protocol	5926aa3	Comptroller/Diamond/facets/PolicyFacet.sol	d79249045d684948cb6cbaeaefe6338f58de8d8b685b2b3889f0c89a44fea77c
⚪ FMV	VenusProtocol/venus-protocol	5926aa3	Tokens/Prime/libs/FixedMath.sol	09b8ce751929c987f98a22a34554ffcad674aa86db8655e889d3421c3608e411
⚪ FPT	VenusProtocol/venus-protocol	5926aa3	Tokens/Prime/libs/FixedMath0x.sol	33d6b5ad05ab3e5724bbee81e7a635eac0fc1b0fb958e159e6d03346b3cf5dca
⚪ SPV	VenusProtocol/venus-protocol	5926aa3	Tokens/Prime/libs/Scores.sol	1bdd8ca4f51535db7270a285e578ade5948b499641c9e48238fb347cdfd98ff8
⚪ IPT	VenusProtocol/venus-protocol	5926aa3	Tokens/Prime/IPrime.sol	5d4c6057c995ffe9c4a53a8b750f5bfe88abeef26ef0734a42852cd3e5be992f
⚪ PST	VenusProtocol/venus-protocol	5926aa3	Tokens/Prime/PrimeStorage.sol	aa1b20712e9e9b33c48f11e6a6e516d637e61ee8fb1f20179d98098b8c8e845f

ID	Repo	Commit	File	SHA256 Checksum
XVP	VenusProtocol/venus-protocol	5926aa3	XVSVault/XVSVaultStorage.sol	b63e57a5f65136f8707e1a8c7830d6227b30004f1e59fd30d226dc74568980ea

APPROACH & METHODS | VENUS - PRIME

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

This audit concerns the changes made in files outlined in this PR: https://github.com/VenusProtocol/venus-protocol/pull/196, from commit af7c9afd7ce153778df2b9029b40a8f2a2359eeb to commit 405f9629dd8ad4d17d447034e586c42bd43a2d0d.

The Venus Prime Rewards program is designed to interact with the Venus lending markets and XVSVault. The system provides users with a cash-back incentive for borrow and supply participation in selected boosted markets.

Prime rewards consist of revocable and irrevocable access. Revocable access to Prime functions as follows:

1. A user becomes eligible by staking in the corresponding XVSVault pool for at least 90 consecutive days, with a minimum of 1,000 XVS staked for the period.
2. After this period they can claim one prime token, which automatically makes the user eligible to begin accruing rewards in any market based on their calculated score.
3. If, at any time, the user adjusts their staked XVS such that their active stake no longer meets the minimum threshold, then their prime token is burned, revoking access. If they were partially through the 90 day staked period at the time of decreasing the stake past the threshold, the staked period is reset.

A user may be minted an irrevocable prime token by a privileged account. This token can still be burned by the privileged account, but is not automatically revocable based on a user's XVS stake.

A user’s score for a given market is determined by the formula:

$$\mathrm{xvs}^{\alpha} \cdot \mathrm{capital}^{1-\alpha}$$

Where xvs represents the total XVS staked by the user and $\mathrm{capital}$ is the total sum of supplies and borrows in the market for the user. A user's score is compared against all other valid participants’ scores in order to determine the proportion of rewards the user receives for the period of time they are active in the rewards program.

A user i’s rewards r for a given time period, in a given market, are calculated via the following formula, which is adapted from the Cobb-Douglas function:

$$r(i) = (t \cdot \mu + l) \frac{T(i)^{\alpha} \cdot V(i)^{1-\alpha}}{\sum_{j \in N} \left(T(j)^{\alpha} \cdot V(j)^{1-\alpha}\right)}$$

Where t is the total income generated by the market during the considered time period, μ is the percent of income distributed as boosted yields, l is bootstrapped liquidity for the time period, $T(i)$ is the total XVS staked by user i during the time period, $V(i)$ is the sum of the amount borrowed and supplied (their *capital*) by user i in the market, α is the *amplification weight*, and N is the set of all active participants considered for the time period. The amplification weight α is a ratio between 0 and 1, and determines the importance each of the XVS staked and capital in a market plays in the score of each user. If α is larger than 0.5, then overall, the user’s XVS stake plays a larger role in determining their score. If α is less than 0.5, then overall, the user’s capital in a market plays a larger role in determining their score. The team states that as a default, α will be set at 0.5 to start.

Any time a user makes an adjustment to their XVS staked or their capital in a given market, it is important that the following two actions are performed sequentially: accruing previous rewards since the last update and then updating the score based on the changes made. This order of actions ensures that rewards are correctly proportionally distributed to participants in the Prime program while ensuring all changes are accurately reflected in the proceeding blocks.

Rewards are distributed from two distinct sources: part of the reward liquidity is from the accumulated reserves of the corresponding market; the other portion is bootstrapped through the contract PrimeLiquidityProvider. The portion of a market’s reserves dedicated to funding the Prime Rewards program is set by governance and can be adjusted.

DEPENDENCIES | VENUS - PRIME

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:

- ERC20 Tokens
- Oracles

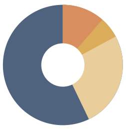
The scope of the audit treats third party entities as black boxes and assumes their functional correctness. However, in the real world, third parties can be compromised and this may lead to lost or stolen assets. Moreover, updates to the state of a project contract that are dependent on the read of the 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.

During the process of the audit, it was assumed that the only tokens distributed as rewards within the `Prime` program are `BTC` , `ETH` , `USDC` and `USDT` , based upon documentation provided by the team. The behavior of the protocol outside this assumption is not addressed within the audit report.

FINDINGS | VENUS - PRIME



35	0	4	2	9	20
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Venus - Prime. Through this audit, we have uncovered 35 issues ranging from different severity levels. Utilizing the techniques of Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
PLP-01	Incorrect Formula May Cause `releaseFunds()` To Revert	Logical Issue	Major	● Resolved
PTP-01	Inconsistent Updates Between `tokenAmountAccrued` And `unreleasedPLPIncome`	Inconsistency	Major	● Resolved
VPB-02	**Centralized Control Of Contract Upgrade**	**Centralization**	**Major**	● **Mitigated**
VPU-02	**Centralization Related Risks**	**Centralization**	**Major**	● **Mitigated**
PLP-03	`sweepToken()` Uses `balance` Instead Of Input `amount_`	Logical Issue	Medium	● Resolved
VPB-04	Misallocation Of Rewards Through Out-Of-Sequence Calls	Concurrency, Volatile Code	Medium	● Resolved
PLP-07	Unprotected Initializer	Coding Issue	Minor	● Resolved
PLP-08	Checks Effects Interactions Pattern Violated	Concurrency	Minor	● Resolved
PPT-14	Potential Out-Of-Gas Exception	Logical Issue	Minor	● Resolved
PPT-15	`calculateAPR()` Does Not Update Oracle	Logical Issue	Minor	● Acknowledged

ID	Title	Category	Severity	Status
PPV-07	Discussion On Unchecked Blocks	Logical Issue	Minor	Resolved
PTP-04	Potential Locked Tokens	Logical Issue	Minor	Acknowledged
SPT-01	Differing Underlying Decimals Causes Varying Behavior From Score Equation	Logical Issue	Minor	Resolved
SPV-01	Potential Inconsistency With Formula	Logical Issue	Minor	Resolved
VPU-03	Missing Input Validation	Volatile Code	Minor	Partially Resolved
CSV-01	Upgrade Sequence Handling	Logical Issue	Informational	Resolved
FMT-01	Fixed Math Library Inconsistencies	Logical Issue	Informational	Acknowledged
PLT-03	Functions Not Included In `IPrimeLiquidityProvider` Interface	Coding Style	Informational	Resolved
PPT-02	Interfaces Can Be Placed In Separate File	Coding Style	Informational	Resolved
PPT-03	Inconsistent Custom Error Usage	Inconsistency	Informational	Resolved
PPT-04	Missing Emit Events	Coding Style	Informational	Resolved
PPT-05	Privileged Actions And Updates To Scores	Logical Issue	Informational	Resolved
PPT-06	State Variable Shadowing	Coding Style	Informational	Resolved
PPT-18	Input `user` Is Not Used	Coding Style	Informational	Resolved
PPT-19	Function Can Be Specified As View	Coding Style	Informational	Resolved

ID	Title	Category	Severity	Status
PPV-01	Single Comptroller Does Not Allow Isolated Pools And Core Pool Handling	Logical Issue	Informational	● Acknowledged
PPV-02	Unused Internal Function	Inconsistency	Informational	● Resolved
PPV-03	Unused Errors	Coding Style	Informational	● Resolved
PPV-08	`delete` Keyword Can Be Used In Place Of Setting Value To Zero	Inconsistency	Informational	● Resolved
PPV-09	Use Negation To Check Nonzero Value	Coding Style	Informational	● Resolved
PTP-06	Implementation Does Not Meet Specification	Inconsistency	Informational	● Resolved
PTP-08	Specific Imports Not Consistently Used	Inconsistency	Informational	● Resolved
VPH-01	Potential For Reentrancy Of Protocol	Concurrency	Informational	● Resolved
VPU-04	Typos And Inconsistencies	Inconsistency	Informational	● Resolved
VPU-05	Missing And Incomplete NatSpec Comments	Logical Issue	Informational	● Resolved

PLP-01 | INCORRECT FORMULA MAY CAUSE `releaseFunds()` TO REVERT

Category	Severity	Location	Status
Logical Issue	● Major	Tokens/Prime/PrimeLiquidityProvider.sol (base): 229	● Resolved

Description

The amount of `tokenAccrued` should be the amount accrued since the update as `tokenAmountAccrued[token_]` is incremented by this value, so that it should only take into account the balance that exceeds the current value of `tokenAmountAccrued[token_]` . Otherwise, the accrued amount can exceed the contract's balance and cause `releaseFunds()` to revert. This can prevent users from claiming their prime interest in the case that the prime contract does not hold enough balance after releasing funds from the `ProtocolShareReserve` and attempts to release funds from the `PrimeLiquidityProvider` .

Scenario

Assume `tokenAmountAccrued[token_] = 100` , `balance = 101` , `deltaBlocks = 100` , and `distributionSpeed = 2` .

Then we have that `accruedSinceUpdate = 200` , so that `balance < accruedSinceUpdate` and thus `tokenAccrued = 101` . Then `tokenAmountAccrued[token_]+= 101` so that `tokenAmountAccrued[token_] = 201` . When there are only 101 tokens in the contract.

Thus when `releaseFunds()` is called, it will attempt to transfer `201` tokens which is greater than the balance of the contract causing a revert.

Recommendation

We recommend using the value of `balance - tokenAmountAccrued[token_]` at the cited line to determine the maximum amount that can be accrued.

Alleviation

`[CertiK, 09/25/2023]` : The client made the recommended changes in commit: 33488f9af5e3b5a3ae8347864dd3917e611aa160.

PTP-01 | INCONSISTENT UPDATES BETWEEN `tokenAmountAccrued` AND `unreleasedPLPIncome`

Category	Severity	Location	Status
Inconsistency	● Major	Tokens/Prime/Prime.sol (base): 545~546; Tokens/Prime/PrimeLiquidityProvider.sol (base): 178~179	● Resolved

Description

Function `accrueInterest()` in contract `Prime` assumes that mapping `tokenAmountAccrued` in `PrimeLiquidityProvider` always holds at least the value for an `underlying` token as is held in mapping `unreleasedPLPIncome` . However, function `releaseFunds()` in `PrimeLiquidityProvider` can be called by anyone at any time, which resets `tokenAmountAccrued` for the `underlying` token to 0. This function call will not have any effect on the current value of `unreleasedPLPIncome` for the `underlying` .

Someone can independently call function `releaseFunds()` for the `underlying` token and consequently, the values for `tokenAmountAccrued` will now be less than the value of `unreleasedPLPIncome` for the `underlying` . As a result, function `accrueInterest()` will revert. Since the only way to reset `unreleasedPLPIncome` to 0 is through function `claimInterest()` being called, and since this function must first call `accrueInterest()` , this halts the functioning of the protocol.

Recommendation

We recommend implementing a design similar to that in `ProtocolShareReserves` where a function `updateAssetState()` is called from `releaseFunds()` in order to zero out the mapping `unreleasedPLPIncome`

Alleviation

`[CertiK]` : The client made changes resolving the finding in commit 8029c8cacdefd52a8022017062406abb0ec3ed9e.

A requirement has been added that `releaseFunds()` can only be called by the `prime` contract. The `prime` contract only calls this function in `PrimeLiquidityProvider` in function `_claimInterest()` which releases all currently available funds to the `prime` contract and updates mapping `unreleasedPLPIncome` to 0.

VPB-02 | CENTRALIZED CONTROL OF CONTRACT UPGRADE

Category	Severity	Location	Status
Centralization	● **Major**	**Comptroller/Diamond/facets/PolicyFacet.sol (base): 12~13; Comptroller/Diamond/facets/SetterFacet.sol (base): 15~16; Tokens/Prime/Prime.sol (base): 82~83; Tokens/Prime/PrimeLiquidityProvider.sol (base): 8~9; XVSVault/XVSVault.sol (base): 27~28**	● **Mitigated**

Description

The contracts `Prime` , `PrimeLiquidityProvider` , `XVSVault` and the Diamond Comptroller are upgradeable. The privileged roles of the proxy have the authority to update the implementation contracts.

Any compromise of the privileged account could allow a hacker to exploit this authority, potentially altering the implementation contract pointed to by the proxy and thus executing malicious functionality within the implementation contract. This includes the ability to take all funds held by the contract.

Recommendation

We recommend that the team make efforts to restrict access to the privileged roles of the proxy contract. A strategy of combining a time-lock and a multi-signature (⅔, ⅗) wallet can be used to prevent a single point of failure due to a private key compromise. In addition, the team should be transparent and notify the community in advance whenever they plan to migrate to a new implementation contract.

Here are some feasible short-term and long-term suggestions that would mitigate the potential risk to a different level and suggestions that would permanently fully resolve the risk.

Short Term:

A combination of a time-lock and a multi signature (⅔, ⅗) wallet mitigate the risk by delaying the sensitive operation and avoiding a single point of key management failure.

- A time-lock with reasonable latency, such as 48 hours, for awareness of privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to a private key compromised;
  AND
- A medium/blog link for sharing the time-lock contract and multi-signers addresses information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the deployed time-lock address.

- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.
- Provide a link to the **medium/blog** with all of the above information included.

Long Term:

A combination of a time-lock on the contract upgrade operation and a DAO for controlling the upgrade operation mitigate the contract upgrade risk by applying transparency and decentralization.

- A time-lock with reasonable latency, such as 48 hours, for community awareness of privileged operations;
  AND
- Introduction of a DAO, governance, or voting module to increase decentralization, transparency, and user involvement;
  AND
- A medium/blog link for sharing the time-lock contract, multi-signers addresses, and DAO information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the deployed time-lock address.
- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.
- Provide a link to the **medium/blog** with all of the above information included.

Permanent:

Renouncing ownership of the `admin` account or removing the upgrade functionality can *fully* resolve the risk.

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

Alleviation

`[Venus, 09/21/2023]` : The admin of these contracts was or will be transferred to 0x939bd8d64c0a9583a7dcea9933f7b21697ab6396, which is the Timelock contract used to execute the normal Venus Improvement Proposals (VIP).

For normal VIPs, the time config is: 24 hours voting + 48 hours delay before the execution.

So, these contracts will be upgraded only via a Normal VIP, involving the community in the process.

`[CertiK, 09/25/2023]` : Considering these steps we have marked this finding as *mitigated*. While this strategy has indeed reduced the risk, it's crucial to note that it has not completely eliminated it. We strongly recommend the team and community to constantly monitor these privileges.

VPU-02 | CENTRALIZATION RELATED RISKS

Category	Severity	Location	Status
Centralization	● **Major**	**XVSVault/XVSVault.sol (update3): 875; Comptroller/Diamond/facets/SetterFacet.sol (base): 506~507; Tokens/Prime/Prime.sol (base): 162, 180, 197, 218, 231, 296, 850; Tokens/Prime/PrimeLiquidityProvider.sol (base): 107, 123, 133, 145, 165, 201; XVSVault/XVSVault.sol (base): 873~874**	● **Mitigated**

Description

Only privileged functions introduced or affected by this PR: https://github.com/VenusProtocol/venus-protocol/pull/196 are in the scope of this audit. All other privileged functions not cited in this finding were not considered and we recommend users carefully review them.

PrimeLiquidityProvider

In the contract `PrimeLiquidityProvider` , the role `_owner` has authority over the following functions:

- `initializeTokens()`
- `setPrimeToken()`
- `sweepToken()`

Any compromise to the `_owner` account may allow a hacker to take advantage of this authority and do the following:

- Initialize tokens that were not planned to be initialized.
- Set the `prime` variable as an account they control, so all funds in the contract can be sent to them.
- Directly remove all funds from the contract immediately via the function `sweepToken()` .

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

- `pauseFundsTransfer()`
- `resumeFundsTransfer()`
- `setTokensDistributionSpeed()`

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

- Prevent or enable the flow of funds into the `prime` contract.

- Change the distribution speed so that the flow of funds to the `prime` contract is faster or slower than intended.

Prime

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

- `updateAlpha()`
- `updateMultipliers()`
- `addMarket()`
- `setLimit()`
- `issue()`
- `burn()`
- `togglePause()`
- `setStakedAt()`

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

- Update alpha or the borrow/supply multipliers to unintended values.
- Add an unintended market.
- Set the limit for revocable and irrevocable tokens so that users can no longer mint new prime tokens. Conversely, they could update the limit to allow more tokens to be minted.
- Issue revocable or irrevocable tokens to themselves.
- Burn any user's revocable or irrevocable tokens.
- Pause or unpause the reward claiming function `claimInterest()`.
- Update the staked at timestamp for any user.

SetterFacet

In the contract `SetterFacet`, the role `admin` has authority over the following functions:

- `_setPrimeToken()`

Any compromise to the `admin` account may allow a hacker to take advantage of this authority and do the following:

- change the `prime` address to a contract with malicious logic, interacting with the `VToken` market contract or `Comptroller` unpredictably

XVSVault

In the contract `XVSVault`, the `admin` role has access to call the following functions:

- `setPrimeToken()`

Any compromise to the `admin` role may allow a hacker to take advantage of this authority and change the `primeToken` address to a contract with malicious logic.

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, 09/21/2023]` :

PrimeLiquidityProvider

The _owner will be set to the Normal Timelock contract, so the mentioned functions will be executable only with the approval of the community.

Regarding the DEFAULT_ADMIN_ROLE, we'll use the AccessControlManager (ACM) deployed at https://bscscan.com/address/0x4788629abc6cfca10f9f969efdeaa1cf70c23555. In this ACM, only 0x939bd8d64c0a9583a7dcea9933f7b21697ab6396 (Normal Timelock) has the DEFAULT_ADMIN_ROLE. And this contract is a Timelock contract used during the Venus Improvement Proposals. We'll allow Normal, Fast-track and Critical timelock contracts to execute the mentioned functions.

Prime

We'll allow Normal, Fast-track and Critical timelock contracts to execute the mentioned functions

SetterFacet

The admin is set to the Normal Timelock contract

XVSVault

We changed that behavior in this commit: https://github.com/VenusProtocol/venus-protocol/commit/860c9598465b0e67092f838e3c5ee2faf7c1b664. Now, only the admin (Normal Timelock contract) will be able to execute this function.

Extra information Current config for the three Timelock contracts:

Normal: 24 hours voting + 48 hours delay Fast-track: 24 hours voting + 6 hours delay Critical: 6 hours voting + 1 hour delay

Addresses of the Timelock contracts:

Normal timelock: https://bscscan.com/address/0x939bD8d64c0A9583A7Dcea9933f7b21697ab6396 Fast-track timelock: https://bscscan.com/address/0x555ba73dB1b006F3f2C7dB7126d6e4343aDBce02 Critical timelock: https://bscscan.com/address/0x213c446ec11e45b15a6E29C1C1b402B8897f606d

`[CertiK, 09/25/2023]` : Considering these steps we have marked this finding as *mitigated*. While this strategy has indeed reduced the risk, it's crucial to note that it has not completely eliminated it. We strongly recommend the team and community to constantly monitor these privileges.

PLP-03 | `sweepToken()` USES `balance` INSTEAD OF INPUT `amount_`

Category	Severity	Location	Status
Logical Issue	● Medium	Tokens/Prime/PrimeLiquidityProvider.sol (base): 207~208	● Resolved

Description

Function `sweepToken()` is meant to transfer an input `amount_` of a specified `token_` to a destination address, from the contract.

Instead, the function transfers the entire `token_.balanceOf(address(this))` to the destination address.

Recommendation

We recommend updating the transfer to send the input `amount_`.

Alleviation

`[CertiK]`: The team made the recommended changes in commit c6b1495f13ad53d8cbf8657f45d56889f45d8800.

VPB-04 MISALLOCATION OF REWARDS THROUGH OUT-OF-SEQUENCE CALLS

Category	Severity	Location	Status
Concurrency, Volatile Code	● Medium	Comptroller/Diamond/facets/PolicyFacet.sol (base): 91~92, 205~206; Tokens/Prime/Prime.sol (base): 509~510, 527~528	● Resolved

Description

It is important for function `executeBoost()` to be called before function `updateScore()` any time an action occurs that affects a user's score. Actions that could affect a user's score include a user updating the stake in the `XVSVault` , updating their position in a supported market, or the team updating how the score gets updated.

If a user can call function `updateScore()` before function `executeBoost()` in one of the scenarios above, then the newly updated score can be applied to the period of time since the last boost was performed. In a case where a user's score drastically increases, this allows them to unfairly gain a larger amount of rewards for the full duration since their last boost.

One such case is outlined below.

Users are transferred the underlying asset via `doTransferOut()` in the `VToken` contract before calling the verify hooks that make calls to `executeBoost()` and `updateScore()` in the `Prime` contract. It is assumed based on context that `VBNB` is a boosted Prime market, and as a consequence, the transfer of funds to users before updating `Prime` allows a user to make calls to `Prime` before the intended calls to `executeBoost()` and `updateScore()` . In particular, a malicious user can take advantage to call the functions of `Prime` in an incorrect order and cause an over-allocation of rewards that allows the user to steal rewards they are not privy to.

For instance, a user can call `updateScore()` before function `executeBoost()` is called in order to apply their new score to their past accumulated difference in `rewardIndex` .

Scenario

User Bob currently has a score of 4 and Alice currently has a score of 1 in the BNB market, which is a boosted market in the `Prime` contract. The `sumOfMembersScore` for the market is 5. We assume for simplicity that the `rewardIndex` for the market and for both users is currently at 0.

1. Ten blocks pass and `accrueInterest()` is called to update the markets `rewardIndex` . We assume it is called directly for simplicity, but in general it can be called via normal actions taken in prime markets. Assume further that now 100 tokens are available as `distributionIncome` . The `delta` for the `rewardIndex` will be 100/5 = 20, so the market now has a `rewardIndex` of 20, and the two users' `rewardIndex` for the market has not yet been updated.
2. Since Bob and Alice were the sole members with active scores for this market for the 10 blocks, it should be that Bob gets 80 tokens and Alice gets 20 tokens from the reward distribution (as Bob had 80% of the total score and Alice

had 20% of the total score).

3. Alice takes out a large borrow in the market, and uses the transfer of BNB in `doTransferOut()` to call function `updateScore()` before function `borrow()` gets to the `borrowVerify()` hook. Assume her score increases and is now 5 so that the `sumOfMemberScore` for the market is now 9. When the borrow function call proceeds to the `borrowVerify()` hook, now `executeBoost()` will be called using Alice's new score, so that she gains (20*5) = 100 tokens instead of her intended 20 tokens.
4. Alice immediately calls `claimInterest()` afterward and takes out the allocated 100 tokens.
5. Now when Bob tries to call `claimInterest()` , he is allocated 80 tokens, but since this is more tokens than are available for release, he will not receive his reward as Alice has effectively stolen it.

Recommendation

We recommend ensuring that functions `executeBoost()` and `updateScore()` cannot be called in an out-of-order sequence in order to apply a new score to a previous boost period.

Additionally, we recommend applying security measures to `VBNB` and any `vToken` which may represent an underlying token with callback features.

One potential solution is to call `executeBoost()` at the beginning of each `VToken` action, and leave `updateScore()` at the end of each `VToken` action. However, note that reentrancy is still possible, and that it should be ensured that no harm can be done to the protocol should a user attempt reentrancy at this level.

Alleviation

`[CertiK, 10/06/2023]` : The client made changes resolving the finding in commit f5e32216191f7959a51d30687df5610af543eca5.

Instead of calling functions `executeBoost()` and `updateScore()` individually in the `prime` contract, each hook now calls a function `accrueInterestAndUpdateScore()` in the `prime` contract.

The functions `executeBoost()` and `updateScore()` no longer exist publicly and are only internally available through the function `accrueInterestAndUpdateScore()` . This prevents a user from calling these functions out-of-sequence.

With regard to the mention of potential reentrancy into other parts of the protocol via the transfer of `BNB` or underlying tokens with hooks, the team states the following:

`[Venus, 10/06/2023]` : For the specific case of VBNB, doTransferOut() uses "transfer", so the available gas in the receiver is limited and probably not enough to perform any reentrancy action.

Code of VBNB: https://github.com/VenusProtocol/venus-protocol/blob/develop/contracts/Tokens/VTokens/VBNB.sol.archive#L147

Regarding tokens implementing hooks on transfer, Venus has a guideline about supported tokens. We don't support ERC777 underlying tokens. But we already have upgradable underlying tokens, so the risk exists in those cases. We maintain conversations with the projects and we assume they won't perform any harmful upgrade.

PLP-07 | UNPROTECTED INITIALIZER

Category	Severity	Location	Status
Coding Issue	● Minor	Tokens/Prime/PrimeLiquidityProvider.sol (base): 79~80	● Resolved

Description

Contract `PrimeLiquidityProvider` does not protect its initializer. An attacker can call the initializer and assume ownership of the logic contract, whereby they can perform privileged operations that trick unsuspecting users into believing that they are the owner of the upgradeable contract.

Recommendation

We recommend calling `_disableInitializers` in the constructor or giving the constructor the `initializer` modifier to prevent the initializer from being called on the logic contract.

Reference: https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable#initializing_the_implementation_contract

Alleviation

`[CertiK, 09/25/2023]`: The client made the recommended changes in commit: 7cb53b6c003933c0e30b0dc4a97bbeeca1b9ebcc.

PLP-08 | CHECKS EFFECTS INTERACTIONS PATTERN VIOLATED

Category	Severity	Location	Status
Concurrency	● Minor	Tokens/Prime/PrimeLiquidityProvider.sol (base): 187~188, 189~190, 207~208, 209~210	● Resolved

Description

PrimeLiquidityProvider.sol

- In function `releaseFunds()` , the event can be emitted before external call to `safeTransfer()` ;
- In function `sweepToken()` , the event can be emitted before external call to `safeTransfer()` ;

Recommendation

We recommend following the checks-effects-interactions pattern.

https://fravoll.github.io/solidity-patterns/checks_effects_interactions.html)

Alleviation

`[CertiK]` : The client made the recommended changes in commit: ed8dd083999212689766e3b5cc4cfa92e8ec24a2.

PPT-14 | POTENTIAL OUT-OF-GAS EXCEPTION

Category	Severity	Location	Status
Logical Issue	● Minor	Tokens/Prime/Prime.sol (base): 167~169, 272, 322, 453, 671	● Resolved

Description

Often, operations loop through all elements of `allMarkets`. If too many markets are added, then these operations may cause transactions to exceed the gas limit.

Recommendation

We recommend either ensuring that there are never so many markets added to `allMarkets` that an out of gas exception will arise during any of the protocols operations, or bounding the length of `allMarkets` to ensure it will not cause such an exception.

Alleviation

`[CertiK, 09/25/2023]`: The client made the recommended changes in commit: 23a95428ee9bcc853593a345f0639a77b2ed483c.

PPT-15 | `calculateAPR()` DOES NOT UPDATE ORACLE

Category	Severity	Location	Status
Logical Issue	● Minor	Tokens/Prime/Prime.sol (base): 796~806	● Acknowledged

Description

The function `calculateAPR()` does not update the oracle for the input `market` or `xvsToken`. Thus the oracle will not update the pivot oracle price before calculating a users capital for their score.

Recommendation

We recommend updating the pivot oracle's prices before using them to calculate a user's capital.

Alleviation

`[Venus, 09/20/2023]`: Issue acknowledged. I won't make any changes for the current version.

_calculateUserAPR is a view function therefore it cannot update price.

We assume that prices will be updated often during other operations. It's a view function, therefore, we consider the security risk to be lower.

Finally, the update in the Resilient Oracle only affects the TWAP oracle, and the Resilient Oracle has a mechanism to revert the transaction if the data is too old.

PPV-07 | DISCUSSION ON UNCHECKED BLOCKS

Category	Severity	Location	Status
Logical Issue	● Minor	Tokens/Prime/Prime.sol (update3): 602~604, 627~629, 730~732, 807~809, 965~967, 1062~1065, 1068~1070, 1074~1076, 1110~1112, 1138~1141, 1179~1181, 1190~1195	● Resolved

Description

The following unchecked blocks are used, however, we do not see an easy way to rationalize that there exists no scenario they will underflow or overflow. If you can please provide the rationale as to why these blocks cannot underflow or overflow, we can check there are no scenarios that were overlooked.

```
  unchecked {
            totalTimeStaked = block.timestamp - userStakedAt;
        }
```

- considering `setStakedAt()` can update `stakedAt` to any value including one larger that the current `block.timestamp`, it is possible that this will underflow.

```

  unchecked {
            supply = (exchangeRate * balanceOfAccount) / EXP_SCALE;
        }
```

- `exchangeRate * balanceOfAccount` may overflow. While it is understood that the `balanceOfAccount` value should be a portion of the `totalSupply` of vTokens used in the `exchangeRate`, if it is possible to manipulate the exchangeRate `to be inconsistently high, the product` exchangeRate * balance` may overflow.

```
  unchecked {
                delta = ((distributionIncome * EXP_SCALE) /
market.sumOfMembersScore);
            }
```

- `distributionIncome * EXP_SCALE` may overflow if `distributionIncome` is large enough.

```
  unchecked {
            supply = (exchangeRate * balanceOfAccount) / EXP_SCALE;
        }
```

- `exchangeRate * balanceOfAccount` may overflow.

```
   unchecked {
            _market.sumOfMembersScore = _market.sumOfMembersScore -
interests[market][user].score + score;
        }
```

- may overflow if `score` is larger than `interests[market][user].score`. Without a check that the addition does not cause overflow, it cannot be ensured that the `_market.sumOfMembersScore` is an invariant.

```
  unchecked {
            supplyUSD = (tokenPrice * supply) / EXP_SCALE;
            borrowUSD = (tokenPrice * borrow) / EXP_SCALE;
        }
```

- `tokenPrice * supply` and `tokenPrice * borrow` may overflow.

```
  unchecked {
                supply = supplyUSD != 0 ? (supply * supplyCapUSD) / supplyUSD : 0;
            }
```

- `supply * supplyCapUSD` may overflow.

```
  unchecked {
                borrow = borrowUSD != 0 ? (borrow * borrowCapUSD) / borrowUSD : 0;
            }
```

- `borrow * borrowCapUSD` may overflow.

```
  unchecked {
            return (index * score) / EXP_SCALE;
        }
```

- `index * score` may overflow.

```
  unchecked {
            return ((((market.totalBorrows() * market.borrowRatePerBlock()) /
EXP_SCALE) *
                market.reserveFactorMantissa()) / EXP_SCALE);
        }
```

- may overflow when multiplying.

```
   unchecked {
            userYearlyIncome = (userScore * _incomeDistributionYearly(vToken)) /
totalScore;
        }
```

- `userScore * _incomeDistributionYearly(vToken)` may overflow.

```
    unchecked {
            userSupplyIncomeYearly = (userYearlyIncome * totalCappedSupply) /
totalCappedValue;
            userBorrowIncomeYearly = (userYearlyIncome * totalCappedBorrow) /
totalCappedValue;
            supplyAPR = totalSupply == 0 ? 0 : ((userSupplyIncomeYearly *
maximumBps) / totalSupply);
            borrowAPR = totalBorrow == 0 ? 0 : ((userBorrowIncomeYearly *
maximumBps) / totalBorrow);
        }
```

- `userYearlyIncome * totalCappedSupply` , `userYearlyIncome * totalCappedBorrow` , `userSupplyIncomeYearly * maximumBps` , and `userBorrowIncomeYearly * maximumBps` may overflow.

Recommendation

We recommend either removing the unchecked blocks or providing the rational behind why these values will not overflow or underflow.

Alleviation

`[CertiK, 11/09/2023]` : The team made the recommended changes in commit 4bcc2f015c58c58f4893c494a47c507682de306a.

PTP-04 | POTENTIAL LOCKED TOKENS

Category	Severity	Location	Status
Logical Issue	● Minor	Tokens/Prime/Prime.sol (base): 557~558; Tokens/Prime/PrimeLiquidityProvider.sol (base): 178~179	● Acknowledged

Description

According to the logic of function `accrueInterest()` , it is possible that some rewards will not be collected by any user. If rewards are currently being issued, but no user has a positive score, then no user can collect these rewards. Even so, all currently unreleased funds issued can be sent from both `PrimeLiquidityProvider` and `ProtocolShareReserve` to the `Prime` contract by anyone. Since no user is privy to these funds, they will become stuck in the contract.

This is the case even if the `Prime` contract calls `releaseFunds()` , since all currently available funds will be released when the function is called.

Recommendation

We recommend including a way to handle funds which are left in the `Prime` contract that no one can collect.

Alleviation

`[Venus, 09/21/2023]` : Issue acknowledged. I won't make any changes for the current version.

We are going to issue Prime tokens at the same time (same transaction) we enable the Prime contracts, so the described scenario will not happen

SPT-01 | DIFFERING UNDERLYING DECIMALS CAUSES VARYING BEHAVIOR FROM SCORE EQUATION

Category	Severity	Location	Status
Logical Issue	● Minor	Tokens/Prime/libs/Scores.sol (base): 29	● Resolved

Description

The input `capital` in `calculateScore` , is an amount of underlying token which may differ in decimals across markets. The same α is used for every market, however, causing the equation to behave differently for each market.

Recommendation

We recommend normalizing the decimals when calculating the score to ensure that the equation is consistent across all supported markets.

Alleviation

`[CertiK, 09/25/2023]` : The client made the recommended changes in commits:

- f1fecacada4bb5d7e4b8cad5b6ca498f2d0d16be;
- 27f84855dc74ebdad9a144103d2db06b0db4ec76.

In addition, they stated they will not support markets with tokens that have more than 18 decimals, so that the normalization equation is well defined for their markets.

SPV-01 | POTENTIAL INCONSISTENCY WITH FORMULA

Category	Severity	Location	Status
Logical Issue	● Minor	Tokens/Prime/libs/Scores.sol (update1): 50	● Resolved

Description

Currently the restrictions on α include the ability for it to be set to 0 or 1. However, this causes potential discrepancies if `xvs` or `capital` is zero.

If $\alpha = 0$:

$$xvs^{\alpha} * capital^{1-\alpha} = capital$$

So that the score should be based solely on the `capital` of the user. However, if `xvs` is zero then the function `calculateScore()` in the `Scores` library will return 0 which may not be consistent with the amount of the user's capital. Note that if a user has an irrevocable prime token, then `xvs` may be 0.

If $\alpha = 1$:

$$xvs^{\alpha} * capital^{1-\alpha} = xvs$$

So that the score should be based solely on the `xvs` of the user. However, if `capital` is zero then `calculateScore()` in the `Scores` library will return 0 which may not be the consistent with the the user's amount of `xvs`.

Recommendation

We recommend ensuring the functions behave as intended and either adjust the logic to handle the cases consistent with the formula or update the comments and documentations to show it behaves as intended in these cases.

Alleviation

`[CertiK, 11/09/2023]`: The made the recommended changes in commits:

- 26fd196485b0e91bd5486fe45a4f5c42dee5d782;
- 64daba3f757085585a705f685e86e468580c0eb5;
- efa219aa33811647bed0ae2ca5100648eb36eb9a

VPU-03 | MISSING INPUT VALIDATION

Category	Severity	Location	Status
Volatile Code	● Minor	Tokens/Prime/Prime.sol (update3): 372~373, 377~378; Tokens/Prime/Prime.sol (base): 139~140, 161~172, 196~197; Tokens/Prime/PrimeLiquidityProvider.sol (base): 144~145, 201~202, 221~222, 227~228, 235~236; Tokens/Prime/libs/Scores.sol (base): 43	● Partially Resolved

Description

PrimeLiquidityProvider.sol

- Function `sweepToken()` is missing a check that the `token_` input is not an initialized token for the contract. This allows the removal of tokens that may be reserved for the `Prime` contract.
- Function `accrueTokens()` is missing a check that the `token_` input is an initialized token. In the contract, initializing tokens is represented as privileged action. However, a user can initialize any token by using its address in the `accrueTokens()` function. This presents a lack of control over when a given token is initialized in the contract.
- Function `setTokenDistributionSpeed()` can be called for token addresses that have not yet been initialized. If a speed is set on an uninitialized token, then anyone can call function `releaseFunds()` for that token and in the call to `accrueTokens()` , the `deltaBlocks` value will be the difference in the current `blockNumber` and 0. If a token balance has been sent to the contract in anticipation of accruing the tokens based on the set speed from its initialized block, then the miscalculated `deltaBlocks` could cause the entire balance to be released to the `prime` contract immediately. Additionally, if the `prime` address is not set before setting distribution speeds, then it may be possible to transfer tokens to the `address(0)` through function `releaseFunds()` , if the initialized token allows transfers to `address(0)` .

Prime.sol

- The specification states alpha must be between 0 and 1 and in the contract `Scores` , it asserts that `alphaNumerator <= alphaDenominator` . The assert statement can be removed from `Scores` and instead the inputs can be checked when they are assigned to ensure they meet the specification. This should be done in the functions `initialize()` and `updateAlpha()` .
- The `initialize()` and `updateAlpha()` functions are missing a check that `alphaDenominator` is set to a nonzero value.
- The function `addMarket()` does not verify that the added `vToken` is a supported market of Venus.
- Function `setStakedAt()` is missing a check that each `timestamps[i]` value is no larger than the current `block.timestamp` . If, for any reason, the privileged role attempts to set a user's `stakedAt` value to a timestamp that is larger than the current `block.timestamp` , a user that does not yet have a prime token can override this

change by making the proper update to their staked xvs, triggering a call to `xvsUpdated()`, setting their `stakedAt` value to the current timestamp. A user who already has a prime token is not affected by the change to `stakedAt`.

Recommendation

We recommend including the checks above.

Alleviation

`[CertiK]`: The client made changes partially resolving the finding in commits

- f7d463b4c9e3467aea026ce33d0f3f643ada3022
- 5a04aa0354c3d5bbf6a91234547920c7f52a65cc
- a7e913ff2bbc4489b8b76b2c01a07151ba0b2cdc
- a8aef0f29f418aa31c570fdff417aeeda47aa436.

However, the team states they opt to acknowledge the recommendation regarding input validation for function `sweepToken()` without making changes, citing they prefer to keep the flexibility within the privileged function.

CSV-01 | UPGRADE SEQUENCE HANDLING

Category	Severity	Location	Status
Logical Issue	● Informational	Comptroller/ComptrollerStorage.sol (update3): 242~243, 263~264	● Resolved

Description

The perceived pattern for upgrade handling is that a new child contract for comptroller storage versions is created each time an upgrade is to take place. We note that the comptroller has already been upgraded to the diamond proxy pattern on-chain at proxy address 0xfD36E2c2a6789Db23113685031d7F16329158384 with corresponding implementation 0xAd69AA3811fE0EE7dBd4e25C4bae40e6422c76C8. Since `ComptrollerV13Storage` is already in use, we recommend moving the addition of the `prime` variable to a `ComptrollerV14Storage` contract, to distinguish the new upgrade.

Recommendation

We recommend moving the addition of the `prime` variable to a `ComptrollerV14Storage` contract, to distinguish the new upgrade.

Alleviation

`[CertiK, 11/09/2023]`: The team made the recommended changes in commit 4eac8359e3364df5898cb4b85b17f6f4c1f71b65.

FMT-01 | FIXED MATH LIBRARY INCONSISTENCIES

Category	Severity	Location	Status
Logical Issue	● Informational	Tokens/Prime/libs/FixedMath0x.sol (base): 1~262	● Acknowledged

Description

The `FixedMath0x` library does not provide clear documentation to completely verify their logic.

For example the comment:

" `x` is now our residual in the range of 1 <= x <= 2 (or close enough)."

does not give clear bounds on `x`. If `x` exceeds 2, then this can lead to potential issues as the Taylor series only converges and thus gives a valid approximation for $0 < x \leq 2$.

In addition, the hexadecimal values chosen are not always the closest value to those in the comments. For example:

```
        // e ^ -32
        if (x <=
int256(0x00000000000000000000000000000000000000000001c8464f76164760000000)) {
            r -=
int256(0x0000000000000000000000000000000100000000000000000000000000000000); // - 32
            x = (x * FIXED_1) /
int256(0x00000000000000000000000000000000000000000001c8464f76164760000000); // / e ^
-32
        }
```

Here the comment gives that the hexadecimal `0x00000000000000000000000000000000000000000001c8464f76164760000000` should represent `e^(-32)`. However this represented as a fixed point number should be

`e^(-32) * 2^(127) = 2154696114062186216855968.0713581760...`

Which rounds down to `2154696114062186216855968`, represented by the hexadecimal `0x00000000000000000000000000000000000000000001C8464F76164681E299A0`.

Furthermore, there is a minimum value set that will behave as the default for small enough inputs. This is the value `-63.875`, but the true minimum that can be reached is approximately -88. Thus users can borrow/supply the minimum amount required to ensure that their ratio is exactly 1 (that is 1/2^127). This would mean that the natural log should return -88, but it returns the default -63.875 instead, allowing the user to slightly game the system.

Currently, the formulas are rewritten using base e, however, we would like to note that they can be written in base 2. The use of base e is optimal for the current library as a Taylor series is used to approximate the value, however, if other approximation methods are used base 2 may be the better choice.

Recommendation

We recommend considering the best balance between gas optimization and precision necessary. Some references to consider when making the choice of fixed math library are listed below. In addition, we recommend ensuring that proper documentation is provided to clarify the design choices of the library and to provide clear bounds on the potential error of the functions.

References:

- From PRB: https://github.com/PaulRBerg/prb-math/tree/v1.0.3;
- From ABDK: https://github.com/abdk-consulting/abdk-libraries-solidity/blob/master/ABDKMath64x64.sol;
- From UniswapV3 (See log2 implementation in TickMath.sol): https://github.com/Uniswap/v3-core/blob/main/contracts/libraries/TickMath.sol

Alleviation

`[Venus, 10/06/2023]` :

1. Regarding the hexadecimal values considered for the different thresholds, we think the root cause is the precision used to calculate those values. Using Octave (with 64-bits float) the obtained values are almost similar to those used in the library.

2. Regarding the minimum value returned, we prefer to keep it as it is. It's not easy to get a ratio of 1/2^127. The borrowed amount is increased with the interest, so, any recalculation of the score would modify it moreover, it would force the user to keep the same amount of XVS staked and QVL. The benefit that this action could generate for the user can be ignored if we compare it with the benefit of increasing the XVS staked or the QVL. Finally, this calculation is used for the scores, not for rewards amounts. Therefore overall it only causes (potentially) a minor unfair reward distribution.

3. Regarding the base e, the Taylor series seems good enough for the precision we need. So, we prefer to keep it as it is.

PLT-03 | FUNCTIONS NOT INCLUDED IN `IPrimeLiquidityProvider` INTERFACE

Category	Severity	Location	Status
Coding Style	● Informational	Tokens/Prime/PrimeLiquidityProvider.sol (update3): 216~217, 256~257	● Resolved

Description

Functions `setMaxTokensDistributionSpeed()` and `setMaxLoopsLimit()` are not included in the `IPrimeLiquidityProvider` interface; all other external-facing, non-initializing functions present in the `PrimeLiquidityProvider` contract are first declared in the interface.

Recommendation

We recommend adding the newly included functions to the `IPrimeLiquidityProvider` interface.

Alleviation

`[CertiK, 11/09/2023]` : The team made the recommended changes in commit 14cca06d676c85f16508bf203bef956f0642d3a8.

PPT-02 | INTERFACES CAN BE PLACED IN SEPARATE FILE

Category	Severity	Location	Status
Coding Style	● Informational	Tokens/Prime/Prime.sol (base): 11~69	● Resolved

Description

The file `Prime.sol` includes several interfaces along with the contract `Prime`.

Recommendation

We recommend putting the interfaces in a separate file and importing them to enhance the readability of the codebase.

Alleviation

`[CertiK, 09/25/2023]`: The client made the recommended changes in commits:

- 08229e60c7a540b8832f32ccfb39af5464642d94;
- aafdacfe848e9b97e10830708b398d1b982fd41d.

PPT-03 | INCONSISTENT CUSTOM ERROR USAGE

Category	Severity	Location	Status
Inconsistency	● Informational	Tokens/Prime/Prime.sol (base): 108~110	● Resolved

Description

Custom errors are used, however, the `constructor()` still uses `require` statements instead of custom errors.

Recommendation

We recommend using custom errors in the `constructor()` to remain consistent.

Alleviation

`[CertiK, 09/25/2023]` : The client made the recommended changes in commit: 08229e60c7a540b8832f32ccfb39af5464642d94.

PPT-04 | MISSING EMIT EVENTS

Category	Severity	Location	Status
Coding Style	● Informational	Tokens/Prime/Prime.sol (base): 161~162, 179~180, 196~197, 217~218, 603~604	● Resolved

Description

Functions that update state variables or perform privileged actions should emit relevant events as notifications.

Recommendation

We recommend adding events for privileged, state-changing actions, and emitting them in their relevant functions.

Alleviation

`[CertiK, 09/25/2023]` : The client made the recommended changes in commit: f73192bd72ecb01fb4bab849e1dcba48c2fe5320.

PPT-05 | PRIVILEGED ACTIONS AND UPDATES TO SCORES

Category	Severity	Location	Status
Logical Issue	● Informational	Tokens/Prime/Prime.sol (base): 171~172, 187~188, 209~210, 660~661	● Resolved

Description

Any time functions `updateAlpha()` , `updateMultipliers()` or `addMarket()` are used to make an update to the protocol, there is a period of time between when the corresponding values are updated, and when all users' scores are updated to incorporate the change. A user can wait for such a change and take advantage of the discrepancy to potentially do one of the following items:

- A new market is added and no one is included in the `sumOfMembersScore` yet. A user updates their own score as soon as the new market is added and for a period of time, they are currently 100% of the receivers for the market rewards.
- The α value is changed to drastically increase most users' scores. One user updates their score while all other scores are the same value. As a result, the updated user is privy to more rewards than others for a period of time. A similar issue could occur with the borrow and supply multipliers.

Recommendation

Some issues could be avoided by including a bool for new markets that keeps rewards from accumulating in `Prime` until all scores have been updated and a privileged account updates the bool.

For updates to α and the borrow and supply multipliers, we recommend either:

1. Refactoring the logic of the contract to support the updates to these values without allowing users to gain an advantage
2. Updating these values by small enough increments over time so that one user cannot gain an advantage through the update of these values. For instance, if α needs to be updated from 0.5 to 0.6, the team can find a small enough increment, e.g. 0.01, and periodically increase the value until the new target value is reached.

Alleviation

`[CertiK 09/26/2023]` : The client made changes in commit f5e32216191f7959a51d30687df5610af543eca5.

The change ensures that previous rewards are collected through `accrueInterest()` before updating users' scores to avoid inaccurate distributions of rewards, especially during updates made to the state variables described above. In particular, this reduces the potential damage that could occur from a user having the change in score applied to the entire duration since they last accrued rewards, which could potentially be much longer than when the update to scoring variables occurred.

The change does not prevent a user from calling `accrueInterestAndUpdateScore()` independently before the team calls function `updateScores()` . For a period of time, the user acting independently may accrue rewards at a greater advantage to other users by being the first and only user to update.

Can you please provide us with the estimated number of blocks the team estimates it will take each time to complete calling `updateScores()` on all active participants.

`[CertiK, 10/12/2023]` : The team states they expect to invoke `updateScores()` for every user in less than 10 minutes (200 blocks in BNB chain).

PPT-06 | STATE VARIABLE SHADOWING

Category	Severity	Location	Status
Coding Style	● Informational	Tokens/Prime/Prime.sol (base): 532~533	● Resolved

Description

The state variable `primeLiquidityProvider` is being shadowed by a local variable in function `accrueInterest()` .

Recommendation

We recommend renaming the local variable to avoid conflicts.

Alleviation

`[CertiK, 09/22/2023]` : The client made the recommended changes in commit: 2c886b65f5546c21226af6e5fe273c776148f6e0.

PPT-18 | INPUT `user` IS NOT USED

Category	Severity	Location	Status
Coding Style	● Informational	Tokens/Prime/Prime.sol (base): 759	● Resolved

Description

The input `user` in the function `_calculateUserAPR()` is never used within the function.

Recommendation

We recommend removing the input or adding functionality that uses it.

Alleviation

`[CertiK, 09/22/2023]` : The client made the recommended changes in commit: e88227062060fa5774c943e09358d7ec67d86f9a.

PPT-19 | FUNCTION CAN BE SPECIFIED AS VIEW

Category	Severity	Location	Status
Coding Style	● Informational	Tokens/Prime/Prime.sol (base): 575~580	● Resolved

Description

The function `_interestAccrued` does not modify any states and can be marked as `view`.

Recommendation

We recommend specifying `_interestAccrued` as a `view` function to ensure it is clear that it does not make state changes.

Alleviation

`[CertiK, 09/22/2023]`: The client made the recommended changes in commit: 70ddd0e3b085b9e032437428079f7fe2c55700b6.

PPV-01 | SINGLE COMPTROLLER DOES NOT ALLOW ISOLATED POOLS AND CORE POOL HANDLING

Category	Severity	Location	Status
Logical Issue	● Informational	Tokens/Prime/Prime.sol (update3): 402~403, 413~414	● Acknowledged

Description

Only a single comptroller is used to determine if a market exists, so the `Prime` contract cannot handle both the core pools and isolated pools in the prime program. While all pools may have a portion of their revenue captured for prime rewards, the only way to become privy to these rewards through the prime program would be to participate in the core pool.

Considering the new design intent that all markets, including the isolated pools markets, will contribute to the prime rewards we would like to ensure that the prime programs design intent is to only allow core markets to be added to the prime program.

Recommendation

We recommend confirming whether the design intent is as outlined above.

Alleviation

`[Venus, 11/09/2023]` : "Issue acknowledged. I will fix the issue in the future, which will not be included in this audit engagement."

PPV-02 | UNUSED INTERNAL FUNCTION

Category	Severity	Location	Status
Inconsistency	● Informational	Tokens/Prime/Prime.sol (update3): 1131~1142	● Resolved

Description

Due to refactoring the code so that all income now originates from the `PrimeLiquidityProvider` , the function `_incomePerBlock()` is no longer used or needed.

Recommendation

We recommend removing the unused function.

Alleviation

`[CertiK, 11/09/2023]` : The team made the recommended changes in commit 386929eaf158955dc219603cc21789394c19d20d.

PPV-03 | UNUSED ERRORS

Category	Severity	Location	Status
Coding Style	● Informational	Tokens/Prime/Prime.sol (update3): 115, 118~119	● Resolved

Description

The cited errors are never used within the `Prime` contract codebase.

Recommendation

We recommend removing or implementing these errors.

Alleviation

`[CertiK, 11/09/2023]` : The team made the recommended changes in commit ef28b347fbe142a432657ba1c00db76cd818dec3.

PPV-08 | `delete` KEYWORD CAN BE USED IN PLACE OF SETTING VALUE TO ZERO

Category	Severity	Location	Status
Inconsistency	● Informational	Tokens/Prime/Prime.sol (update3): 500	● Resolved

Description

All other instances of assigning zero have been replaced with using `delete` .

Recommendation

We recommend using `delete` for the remaining instances to be consistent.

Alleviation

`[CertiK, 11/09/2023]` : The team made the recommended changes in commit 5af27488693213cd1fc9a8659a440d33e5c5b634.

PPV-09 | USE NEGATION TO CHECK NONZERO VALUE

Category	Severity	Location	Status
Coding Style	● Informational	Tokens/Prime/Prime.sol (update3): 1008~1009	● Resolved

Description

In function `_updateRoundAfterTokenMinted()` , value `totalScoreUpdatesRequired` is checked to be nonzero by checking if the value is greater than zero. For consistency across the codebase, it can be checked that `totalScoreUpdatesRequired != 0` instead.

Recommendation

We recommend updating the value to check consistently across the codebase.

Alleviation

`[CertiK, 11/09/2023]` : The team made the recommended changes in commit 6581db184c2245cfb590d5b74d54fd94b0ab5e64.

PTP-06 | IMPLEMENTATION DOES NOT MEET SPECIFICATION

Category	Severity	Location	Status
Inconsistency	● Informational	Tokens/Prime/Prime.sol (base): 712~714; Tokens/Prime/PrimeStorage.sol (base): 31~32	● Resolved

Description

- In `PrimeStorage`, constant `MAXIMUM_XVS_CAP` is set as 10000 * EXP_SCALE, however, the documentation provided states that the maximum XVS cap should be 100,000 XVS.

- In `Prime`, the `_incomePerBlock()` uses formula

```
((((market.totalBorrows() * market.borrowRatePerBlock()) / EXP_SCALE) *
market.reserveFactorMantissa()) /
          EXP_SCALE);
```

However, the documentation states that the `_incomePerBlock` should be measured by

```
(borrowRatePerBlock * totalBorrows) - (supplyRatePerBlock*(cash + borrows -
reserves))
```

While the implementation and specification are mathematically equivalent, it is observed that the implementation is more direct and less gas intensive. The implementation also more accurately reflects what occurs within each `VToken` contract.

Recommendation

We recommend updating the `MAXIMUM_XVS_CAP` to be consistent with its intended value.

For the `_incomePerBlock()` we recommend adjusting the documentation to match the implementation in the codebase in order to reflect the meaning of the implementation better.

Alleviation

`[CertiK, 09/22/2023]`: The client updated their documentation and made the recommended changes in commit: 860c9598465b0e67092f838e3c5ee2faf7c1b664.

PTP-08 | SPECIFIC IMPORTS NOT CONSISTENTLY USED

Category	Severity	Location	Status
Inconsistency	● Informational	Tokens/Prime/Prime.sol (base): 3~9; Tokens/Prime/libs/FixedMath.sol (base): 6~7; Tokens/Prime/libs/Scores.sol (base): 5~6	● Resolved

Description

Many of the added files use specific imports, however, some import the entire file.

Recommendation

We recommend using specific imports to clarify what is used and remain consistent.

Alleviation

`[CertiK, 09/22/2023]` : The client made the recommended changes in the following commits:

- b2fa6d549ced74aa8254bb8d367ac240813aabcf;
- adb85352137dd6f2b596527d1133af0d98cece00.

VPH-01 | POTENTIAL FOR REENTRANCY OF PROTOCOL

Category	Severity	Location	Status
Concurrency	● Informational	Comptroller/Diamond/facets/PolicyFacet.sol (update1): 93~94, 164~165; Tokens/Prime/Prime.sol (update1): 350~351	● Resolved

Description

As of commit f5e32216191f7959a51d30687df5610af543eca5 the public functions `executeBoost()` and `updateScore()` in the `Prime` contract have been replaced with a public function `accrueInterestAndUpdateScore()` wherein the internal versions of these functions are called in their correct sequence.

This change resolves finding PFD-01 concerning the misallocation of rewards via reentrancy in the `VBNB` contract, during the borrowing of an asset. In the `VBNB` contract (or if a market is created for an asset with a hook in the future), it is still possible to perform calls to the `Prime` contract when someone borrows or redeems, at the level of `doTransferOut()`, before function `accrueInterestAndUpdateScore()` is called for the user and the market within the `Prime` contract.

The finding is set as informational since no perceived attacks can currently be accomplished via unintended entry into the `Prime` contract before the sequence of intended calls is completed.

Recommendation

It is understood that in this case, `accrueInterestAndUpdateScore()` must be called *after* the `doTransferOut()` function is called, making this potential unintended entry mid-sequence unavoidable in the case of the `VBNB` contract. We recommend ensuring that no harm can be done to the protocol should a user attempt to call into the `Prime` contract at this level.

Alleviation

`[Venus 10/06/2023]` :

For the specific case of VBNB, doTransferOut() uses "transfer", so the available gas in the receiver is limited and probably not enough to perform any reentrancy action.

Code of VBNB: https://github.com/VenusProtocol/venus-protocol/blob/develop/contracts/Tokens/VTokens/VBNB.sol.archive#L147

Regarding tokens implementing hooks on transfer, Venus has a guideline about supported tokens. We don't support ERC777 underlying tokens. But we already have upgradable underlying tokens, so the risk exists in those cases. We maintain conversations with the projects and we assume they won't perform any harmful upgrade.

VPU-04 | TYPOS AND INCONSISTENCIES

Category	Severity	Location	Status
Inconsistency	● Informational	Tokens/Prime/Prime.sol (update3): 568, 751, 1097, 1098; Tokens/Prime/PrimeLiquidityProvider.sol (update3): 84~85; Comptroller/Diamond/facets/PolicyFacet.sol (base): 48, 85, 157, 199, 272, 341, 398; Tokens/Prime/Prime.sol (base): 176~177, 179, 196, 213, 227, 336, 373, 495~497, 510~516, 600, 789~790, 820~821, 847; Tokens/Prime/PrimeLiquidityProvider.sol (base): 65, 72, 119, 129, 174, 198, 270~271, 272~273; Tokens/Prime/PrimeStorage.sol (base): 42~52	● Resolved

Description

In the contract `Prime` :

- The comments above `updateMultipliers()` do not warn that the supply and borrow multiplier should be converted to `1e18` .
- The functions `updateMultipliers()` and `addMarket()` have the same inputs, but do not follow the same conventions.
- The comment above the function `setLimit()` misspells "minted" as "mined".
- The comment above the function `_xvsBalanceOfUser()` for the parameter `user` is not consistent with the function.
- The comment above the function `_xvsBalanceForScore()` misspelled "calculate" as "calcukate".
- In the comment above `togglePause()` , "unpause" is misspelled as "unpuase".
- Functions `calculateAPR()` and `estimateAPR()` should include comments specifying that the returned value is in BPS.
- The comments above the function `issue()` has "is the tokens being issued", which should be "are the tokens being issued".
- The check that a market is a supported prime market or that the user has a prime token is not consistent between the functions `executeBoost()` and `updateScore()` .
- The comments above the function `_claimInterest()` state "the market for which claim", which should be "the market for which to claim".
- The comment above the function `_accrueInterestAndUpdateScore()` includes the word "interest" misspelled as "interes."
- The comments above the function `_interestAccrued()` state "for which calculate", which should be "for which to calculate".

- The comment above the function `claimInterest()` states "the amount of tokens transferred to the user", however, it would be more accurate to say "the amount of tokens transferred to the msg.sender".

In the contract `PrimeLiquidityProvider` :

- The comment above the error `InsufficientBalance` references the `swapRouter` , when it should reference the `PrimeLiquidityProvider` contract.
- The comment above the function `initialize()` references the `RewardsDistributor` , when it should reference the `PrimeLiquidityProvider` contract.
- The comment above the function `pauseFundsTransfer()` states "Emits FundsTransferPaused on success", however, this event is not emitted.
- The comment above the function `resumeFundsTransfer()` states "Emits FundsTransferResumed on success", however, this event is not emitted.
- The comment above the function `releaseFunds()` states "token_ The list of tokens to claim tokens", however, the input is a single `address` and not a list.
- The comment above the function `sweepToken()` states "Throw InsufficientBalance on Zero address(token)", however, it throws `InsufficientBalance` if the input amount of token exceeds the contracts balance.
- The local variable `intializedBlock` contains misspelling "intialized" for the word "initialized".
- The comment above the error `TokenNotInitialized` says "Error thrown when interest accrue is called for not initialized token." The comment may read better as "Error thrown when accrueTokens is called for an uninitialized token."

In the contract `PolicyFacet` :

- The hooks `mintVerify()` , `redeemVerify()` , `borrowVerify()` , `repayBorrowVerify()` , `liquidateBorrowVerify()` , `seizeVerify()` , and `transferVerify()` all include a comment which states that the purpose of the function is to validate each action. Instead, the hooks are now used to update rewards and scores in the `Prime` contract. Additionally, the `VToken` contract includes comments that refer to these functions as "defense hooks" which is not their purpose.

In the contract `PrimeStorage` :

- The public variables `_totalIrrevocable` , `_totalRevocable` , `_revocableLimit` , and `_irrevocableLimit` have a leading underscore when they are external facing variables.

Recommendation

We recommend fixing the typos and inconsistencies mentioned above.

Alleviation

`[CertiK, 10/12/2023]` : The client made the recommended changes in commits:

- 42c565b52fa3a1d43f1df4d7249ddcfc6a9d83a6;
- 7fbe58a77cb97dc5da992ee26593701bfab54f4a;
- 17d5c2ad1e12b2129eda814dee95d4eb6c465e4c
- e1cbc1661853bd5d4c14c824de246e2082e05262
- d5636e0a9fbcf989eee2945be70abc53aa97bd64;
- a96eb1eb9b8b231892c93d6a998d4f4076d035f2

VPU-05 | MISSING AND INCOMPLETE NATSPEC COMMENTS

Category	Severity	Location	Status
Logical Issue	● Informational	Comptroller/Diamond/facets/SetterFacet.sol (update3): 537~541; Tokens/Prime/Prime.sol (base): 71~80, 119~120, 174~178, 477~480, 564~568, 574~575, 592, 644; Tokens/Prime/PrimeLiquidityProvider.sol (base): 161~164, 172~177, 239	● Resolved

Description

In the contract `PrimeLiquidityProvider` :

- The comments above `setPrimeToken()` do not reflect that it emits the `PrimeTokenUpdated` event and that access is restricted to governance by the `onlyOwner` modifier.
- The comments above `releaseFunds()` do not reflect that it throws `FundsTransferIsPaused` if paused.
- The comments above `getBlockNumber()` do not include the return value.

In the contract `Prime` :

- There are no comments reflecting the access restriction of functions, the events emitted, or the errors thrown.
- There are no comments for the defined errors.
- There are no NatSpec comments for the function `initialize()` .
- The comments above `updateMultipliers()` do not include the parameter `market` .
- The comments above `isEligible()` do not include the return value.
- The comments above `getInterestAccrued()` do not include the return value.
- There are no comments for the function `_interestAccrued()` .
- There are no comments for the function `_getUnderlying()` .
- The comments above `claimInterest()` do not include the parameter `user` .

In the contract `SetterFacet`

- The function `_setForcedLiquidation()` does not have a comment stating it allows a privileged role to call it. All other functions that have their access controlled by `ensureAllowed()` have such a comment.

Recommendation

We recommend making adding the NatSpec comments mentioned above.

Alleviation

`[CertiK, 10/12/2023]` : The client made the recommended changes in commits:

- 2825c16b75d90803637a3ac502db5eaeb72ca93f;
- 91b247497b83db509e11146a71e98fcabe843ae7;
- eb36a3fd43e9381e4ad2dd2e4646f84685043577;
- 2abb751c90ca3517141deb1d852a35ebb036070e;
- e1cbc1661853bd5d4c14c824de246e2082e05262.
- 69fe74963b6af7f0cb48c9eac4f3cd0a832d4436.

OPTIMIZATIONS | VENUS - PRIME

ID	Title	Category	Severity	Status
PLP-09	Unused State Variable	Coding Issue	Optimization	● Resolved
PLT-01	Unnecessary Use Of Storage Placeholder	Coding Issue	Optimization	● Resolved
PPT-11	Use Temporary Variable To Save Reading From Storage	Gas Optimization	Optimization	● Acknowledged
PPT-12	Unnecessary Addition	Gas Optimization	Optimization	● Resolved
PPT-13	`for` Loop Optimization	Gas Optimization	Optimization	● Resolved
PPT-20	Inefficient Check	Code Optimization	Optimization	● Resolved
PPV-04	Array Length Can Be Cached Earlier	Code Optimization	Optimization	● Resolved
PPV-06	Unnecessary Initialization	Code Optimization	Optimization	● Resolved

PLP-09 | UNUSED STATE VARIABLE

Category	Severity	Location	Status
Coding Issue	● Optimization	Tokens/Prime/PrimeLiquidityProvider.sol (update1): 15	● Resolved

Description

Some state variables are not used in the codebase.

Variable `EXP_SCALE` in `PrimeLiquidityProvider` is never used in `PrimeLiquidityProvider` .

```
15      uint256 internal constant EXP_SCALE = 1e18;
```

```
 8  contract PrimeLiquidityProvider is AccessControlledV8, PausableUpgradeable {
```

Recommendation

We recommend ensuring that all necessary state variables are used and redundant variables are removed.

Alleviation

`[CertiK, 10/12/2023]` : The client made the recommended changes in commit: e94a190b220e40e572c939c680dc92e166a7acb9.

PLT-01 | UNNECESSARY USE OF STORAGE PLACEHOLDER

Category	Severity	Location	Status
Coding Issue	● Optimization	Tokens/Prime/PrimeLiquidityProvider.sol (update3): 43~44	● Resolved

Description

It is understood that the `PrimeLiquidityProvider` contract is an upgradeable child contract. Being the case, a storage placeholder `__gap` variable is unneeded, since all new variables can be appended to the end in the case of an upgrade.

Recommendation

If `PrimeLiquidityProvider` is to remain a child contract, we recommend removing the unneeded `__gap` placeholder variable.

Alleviation

`[CertiK, 11/09/2023]` : The team made the recommended change in commit 425381df43e48c0603a73b722c763dbf1d33db15.

PPT-11 | USE TEMPORARY VARIABLE TO SAVE READING FROM STORAGE

Category	Severity	Location	Status
Gas Optimization	● Optimization	Tokens/Prime/Prime.sol (base): 167, 272, 273~274, 322, 453, 454~460, 671	● Acknowledged

Description

Often storage variables are read multiple times, when they can be stored as a temporary variable to reduce the amount of times it is read from storage.

Recommendation

We recommend using a temporary variable to store the storage variable.

Alleviation

`[Venus, 09/21/2023]` : Issue acknowledged. I won't make any changes for the current version.

PPT-12 | UNNECESSARY ADDITION

Category	Severity	Location	Status
Gas Optimization	● Optimization	Tokens/Prime/Prime.sol (base): 737	● Resolved

Description

The function `_incomeDistributionYearly()` has the following implementation:

```
function _incomeDistributionYearly(address vToken) internal view returns (uint256
amount) {
        uint256 totalIncomePerBlockFromMarket = _incomePerBlock(vToken);
        uint256 incomePerBlockForDistributionFromMarket =
(totalIncomePerBlockFromMarket * _distributionPercentage()) /
            IProtocolShareReserve(protocolShareReserve).MAX_PERCENT();
        amount += BLOCKS_PER_YEAR * incomePerBlockForDistributionFromMarket;

        uint256 totalIncomePerBlockFromPLP =
IPrimeLiquidityProvider(primeLiquidityProvider)
            .getEffectiveDistributionSpeed(_getUnderlying(vToken));
        amount += BLOCKS_PER_YEAR * totalIncomePerBlockFromPLP;
    }
```

However, the first time `amount` is updated it can be set equal to `BLOCKS_PER_YEAR * incomePerBlockForDistributionFromMarket` as the `amount` will be zero, saving an unnecessary addition operation.

Recommendation

We recommend removing the unnecessary addition operation.

Alleviation

`[CertiK, 09/22/2023]` : The client made the recommended changes in commit: e9bbe5a807406cde6e1ed4bd040b862bd6923764.

PPT-13 | `for` LOOP OPTIMIZATION

Category	Severity	Location	Status
Gas Optimization	● Optimization	Tokens/Prime/Prime.sol (base): 167~172, 234, 239, 272, 322, 453, 664, 671	● Resolved

Description

In general, the counter in a for loop can be incremented or decremented in an unchecked block as it cannot overflow or underflow, saving gas as it will not perform a check for overflow or underflow.

Additionally, it saves a small amount of gas to increment an index in a `for` loop from the left instead of from the right side as it performs fewer operations.

Recommendation

We recommend incrementing the index of the for loop in an unchecked block with a prefix increment.

Alleviation

`[CertiK, 10/12/2023]` : The client made the recommended changes in commits:

- c7fa933b971b52418f1d10122e5024562d89e8c4;
- e1cbc1661853bd5d4c14c824de246e2082e05262;
- fc6a76e29c6b59a03a9ca8f5b4072aa2b9492fa7;
- e7f211a4283595dec9484afd842afac25f6b43dc.

PPT-20 | INEFFICIENT CHECK

Category	Severity	Location	Status
Code Optimization	● Optimization	Tokens/Prime/Prime.sol (base): 411~417, 773	● Resolved

Description

The `totalCappedValue` is always less than or equal to the `totalValue`. Thus it is sufficient to check that the `totalCapped` value is 0.

Recommendation

We recommend removing the `or` statement checking that `totalValue` is zero.

Alleviation

`[CertiK, 09/22/2023]`: The client made the recommended changes in commit: b5ac2fa398ca21cd4f2052ebb39fa72dfcb33e15.

PPV-04 | ARRAY LENGTH CAN BE CACHED EARLIER

Category	Severity	Location	Status
Code Optimization	● Optimization	Tokens/Prime/Prime.sol (update3): 239~241	● Resolved

Description

In the function `getPendingInterests()` , `allMarkets.length` is cached after defining the `pendingInterests` array. It can be cached prior to defining the `pendingInterests` array to save reading `allMarkets.length` for the length of the array.

Recommendation

We recommend caching `allMarkets.length` before defining the `pendingInterests` array.

Alleviation

`[CertiK, 11/09/2023]` : The team made the recommended changes in commit b20bd467359ac92da62bad6f84643ddeed202f8d.

PPV-06 | UNNECESSARY INITIALIZATION

Category	Severity	Location	Status
Code Optimization	● Optimization	Tokens/Prime/Prime.sol (update3): 376	● Resolved

Description

The `for` -loop in the function `setStakedAt()` initializes `uint256 i = 0` . However, no other `for` -loops in the contract initialize their counter.

Recommendation

We recommend removing the initialization to 0 as its default value is 0.

Alleviation

`[CertiK, 11/09/2023]` : The team made the recommended changes in commit c1dddbcb72e154e8dc84d63e27e039c1aaf30696.

APPENDIX | VENUS - PRIME

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.
Coding Issue	Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues.
Concurrency	Concurrency findings are about issues that cause unexpected or unsafe interleaving of code executions.
Inconsistency	Inconsistency findings refer to different parts of code that are not consistent or code that does not behave according to its specification.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
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.

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.

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.

CertiK | Securing the Web3 World

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.