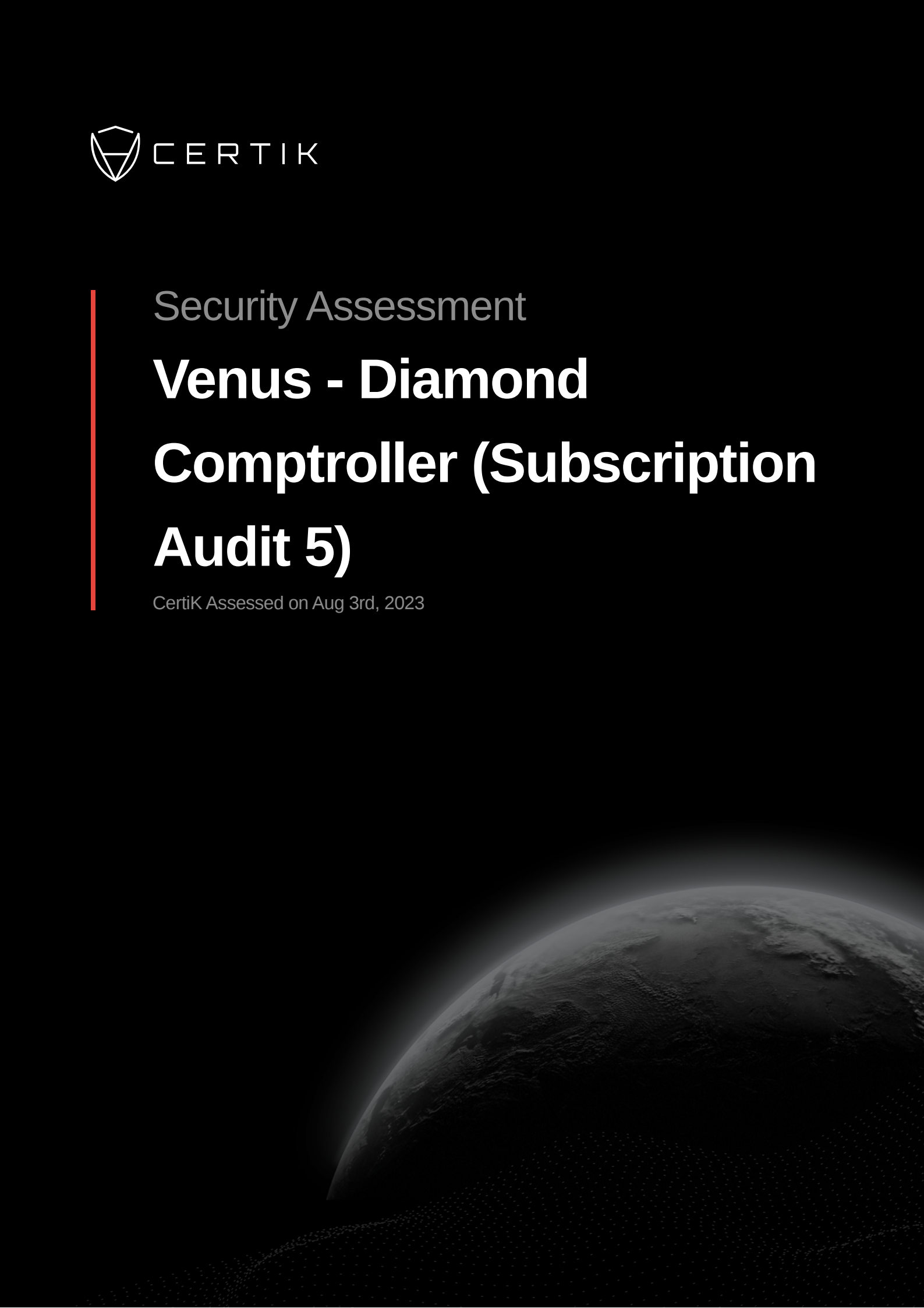
CERTIK

Security Assessment

Venus - Diamond Comptroller (Subscription Audit 5)

CertiK Assessed on Aug 3rd, 2023



CertiK Assessed on Aug 3rd, 2023

Venus - Diamond Comptroller (Subscription Audit 5)

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

Executive Summary

TYPES	ECOSYSTEM	METHODS
DeFi	Binance Smart Chain (BSC)	Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 08/03/2023	N/A

CODEBASE

https://github.com/VenusProtocol/venus-protocol

View All in Codebase Page

COMMITS

base: 94bc2e414e33ebf6c05d35c1605dcbd48fa932f5

update: 7417d8f4b17eb156dd44a8b4d8eb6dbf3e6e4015

View All in Codebase Page

Vulnerability Summary

10	7	2	0	1	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.
2 Major	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.
0 Medium			Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
1 Minor	1 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.
7 Informational	7 Resolved		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 - DIAMOND COMPTROLLER (SUBSCRIPTION AUDIT 5)

CODEBASE | VENUS - DIAMOND COMPTROLLER (SUBSCRIPTION AUDIT 5)

Repository

https://github.com/VenusProtocol/venus-protocol

Commit

base: 94bc2e414e33ebf6c05d35c1605dcbd48fa932f5

update: 7417d8f4b17eb156dd44a8b4d8eb6dbf3e6e4015

AUDIT SCOPE

VENUS - DIAMOND COMPTROLLER (SUBSCRIPTION AUDIT 5)

18 files audited · 1 file with Acknowledged findings · 3 files with Mitigated findings · 5 files with Resolved findings · 9 files without findings

ID	Repo	File	SHA256 Checksum
SFD	VenusProtocol/venus-protocol	contracts/Comptroller/Diamond/facets/SetterFacet.sol	d6171e7cc75667a7a5b9c672d3a5e96ff30421e84539cd1a80cadcd1db53f85c
MFD	VenusProtocol/venus-protocol	contracts/Comptroller/Diamond/facets/MarketFacet.sol	264a75e42404aeb3ad06c420269f9347f44644021c2be904c8c7d9c076ee3907
RFD	VenusProtocol/venus-protocol	contracts/Comptroller/Diamond/facets/RewardFacet.sol	767dc688c7bbe368adf7bcc1e607abdf3ac0b6ba7d92c5b6eaa13fd0ef6e32cf
DDC	VenusProtocol/venus-protocol	contracts/Comptroller/Diamond/Diamond.sol	c2ac8d8c4e2cccc3f70446ed397647bab0bf3c81a6c54ae648210977ca8df317
FBD	VenusProtocol/venus-protocol	contracts/Comptroller/Diamond/facets/FacetBase.sol	a2d9e837eaa10885a5af90fcacd82cdcbebcc95d9e5f1559f38fde78c6e059e9
PFD	VenusProtocol/venus-protocol	contracts/Comptroller/Diamond/facets/PolicyFacet.sol	e9c97497ac5966dd47ad067133355d9b90dce8f22ce0171988a888f9ce9a53f1
XVS	VenusProtocol/venus-protocol	contracts/Comptroller/Diamond/facets/XVSRewardsHelper.sol	39fce7d9432ab2cdaa702b3096e7eab24b4f6623065dc0343139e1f44ede3cfb
CSC	VenusProtocol/venus-protocol	contracts/Comptroller/ComptrollerStorage.sol	b7f23e7ecd1626f98f2f90c76e077d8c0357cd842b96030810a4f00ed25af747
VAI	VenusProtocol/venus-protocol	contracts/Tokens/VAI/VAIController.sol	d5e6337e7b0f84042f27a36ba163dc804e6d5ec9c6dae7938017e6b3acd2a8e0
IDC	VenusProtocol/venus-protocol	contracts/Comptroller/Diamond/interfaces/IDiamondCut.sol	abf621fbd52ac52d50150591e1cd4dcd466f42856aa0eafd3caf16e1c8c8165b
IMF	VenusProtocol/venus-protocol	contracts/Comptroller/Diamond/interfaces/IMarketFacet.sol	b3512f697d9f2a3f48b65a8f82569a6a8d5f0c21c697376988ad2e64fe9940d0
IPF	VenusProtocol/venus-protocol	contracts/Comptroller/Diamond/interfaces/IPolicyFacet.sol	7cc2eb58204d624a41e510a3934bafff2de6c1b6c1cb5027703e1666030c1ef9

ID	Repo	File	SHA256 Checksum
IRF	VenusProtocol/venus-protocol	contracts/Comptroller/Diamond/interfaces/IRewardFacet.sol	1ea472365d9ee2def7a8eb49cfb9a911d091d1635336adb3932baa2ee24695e3
ISF	VenusProtocol/venus-protocol	contracts/Comptroller/Diamond/interfaces/ISetterFacet.sol	325403ba4dea5480b3fdce1cc81c4b543b93c8fa2ac8b16d919a2a5214bbcfc6
CIC	VenusProtocol/venus-protocol	contracts/Comptroller/ComptrollerInterface.sol	c4bc993be50bb114c8e737e126d0ccc2de1444ea8b97d43e364caf266be06367
CLL	VenusProtocol/venus-protocol	contracts/Lens/ComptrollerLens.sol	18827f15ab8f90d167ce9f4f6d78f1631502521e157e8797f2b08262377198cd
SLL	VenusProtocol/venus-protocol	contracts/Lens/SnapshotLens.sol	ff9b2e0be96674ed298789359f4d8a06d109c9bd7ecc63fb8a2ed7ae29d37409
VAC	VenusProtocol/venus-protocol	contracts/Tokens/VAI/VAIControllerStorage.sol	0e0056dce729fbb27e82757f9204e9d78b3a857bafef50b642ef195d8f4c4e05

APPROACH & METHODS

VENUS - DIAMOND COMPTROLLER (SUBSCRIPTION AUDIT 5)

This report has been prepared for Venus to discover issues and vulnerabilities in the source code of the Venus - Diamond Comptroller (Subscription Audit 5) 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.

REVIEW NOTES

VENUS - DIAMOND COMPTROLLER (SUBSCRIPTION AUDIT 5)

The audit primarily concerns the adoption of the multi-facet proxy pattern specified in EIP-2535 as an upgrade in implementation to the already deployed `Unitroller` contract at address https://bscscan.com/address/0xfd36e2c2a6789db23113685031d7f16329158384. At the time of the audit, the implemented logic for this proxy, which is used in consideration of the analysis, is found at address https://bscscan.com/address/0x909dd16b24cef96c7be13065a9a0eaf8a126ffa5.

Additionally, the following files in the audit scope were analyzed as a a delta audit based on changes made in https://github.com/VenusProtocol/venus-protocol/pull/224, rather than a full audit of the logic present in each file.

- ComptrollerInterface.sol
- ComptrollerLens.sol
- SnapshotLens.sol
- VAIController.sol
- VAIControllerStorage.sol

PROJECT ASSUMPTIONS

VENUS - DIAMOND COMPTROLLER (SUBSCRIPTION AUDIT 5)

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

- The Multi-Facet Proxy Pattern (see EIP-2535) implemented within this project is unique in that it allows contracts of any size to be implemented. However, the use of this structure with a currently existing contract using compiler version 0.5.16 allows for the potential of overflow issues that may not have existed otherwise due to the allowance of contracts of any size. Even so, it is reasonable to assume that the Venus Team does not plan to add more than $2^{96} - 1$ selectors for a given facet contract address;
- The interfaces for each facet serve as documentation for the expected association of function signatures with each facet. Since each signature can only be associated to one facet within the Comptroller, and since each facet inherits `FacetBase`, it is assumed that the user-facing functions of `FacetBase` will associated to the `RewardFacet` based on its corresponding interface declarations;

Recommendations

We recommend all assumptions about the behavior of the project are thoroughly reviewed and, if the assumptions do not match the intention of the protocol, documenting the intended behavior for review.

INSPECTION OF STORAGE IN UPGRADED CONTRACTS

VENUS - DIAMOND COMPTROLLER (SUBSCRIPTION AUDIT 5)

Current `Unitroller` (proxy) Deployment:

https://bscscan.com/address/0xfd36e2c2a6789db23113685031d7f16329158384

Current `Comptroller` Implementation in Use by `Unitroller`:

https://bscscan.com/address/0x909dd16b24cef96c7be13065a9a0eaf8a126ffa5

Analysis

The storage layout of the audited file `Diamond.sol` was assessed against the currently existing storage layout held by the `Comptroller` implementation cited above.

From analysis of the tables below, it is confirmed that the `Diamond` contract intended to replace the currently implemented `Comptroller` contract through upgrade of the `Unitroller` cited above retains the same storage layout of the current implementation, and all newly added state variables are held in slots appended to the end of the currently existing storage. It is also noted that all facets inherit the same storage layout. Hence, the upgrade will not cause storage collisions.

Storage Layout of Currently Deployed `Comptroller.sol`

Name	Type	Slot	Offset	Bytes
admin	address	0	0	20
pendingAdmin	address	1	0	20
comptrollerImplementation	address	2	0	20
pendingComptrollerImplementation	address	3	0	20
oracle	contract PriceOracle	4	0	20
closeFactorMantissa	uint256	5	0	32
liquidationIncentiveMantissa	uint256	6	0	32
maxAssets	uint256	7	0	32

Name	Type	Slot	Offset	Bytes
accountAssets	mapping(address => contract VToken[])	8	0	32
markets	mapping(address => struct ComptrollerV1Storage.Market)	9	0	32
pauseGuardian	address	10	0	20
_mintGuardianPaused	bool	10	20	1
_borrowGuardianPaused	bool	10	21	1
transferGuardianPaused	bool	10	22	1
seizeGuardianPaused	bool	10	23	1
mintGuardianPaused	mapping(address => bool)	11	0	32
borrowGuardianPaused	mapping(address => bool)	12	0	32
allMarkets	contract VToken[]	13	0	32
venusRate	uint256	14	0	32
venusSpeeds	mapping(address => uint256)	15	0	32
venusSupplyState	mapping(address => struct ComptrollerV1Storage.VenusMarketState)	16	0	32
venusBorrowState	mapping(address => struct ComptrollerV1Storage.VenusMarketState)	17	0	32
venusSupplierIndex	mapping(address =>	18	0	32

Name	Type	Slot	Offset	Bytes
	mapping(address => uint256))			
venusBorrowerIndex	mapping(address => mapping(address => uint256))	19	0	32
venusAccrued	mapping(address => uint256)	20	0	32
vaiController	contract VAIControllerInterface	21	0	20
mintedVAIs	mapping(address => uint256)	22	0	32
vaiMintRate	uint256	23	0	32
mintVAIGuardianPaused	bool	24	0	1
repayVAIGuardianPaused	bool	24	1	1
protocolPaused	bool	24	2	1
venusVAIRate	uint256	25	0	32
venusVAIVaultRate	uint256	26	0	32
vaiVaultAddress	address	27	0	20
releaseStartBlock	uint256	28	0	32
minReleaseAmount	uint256	29	0	32
borrowCapGuardian	address	30	0	20
borrowCaps	mapping(address => uint256)	31	0	32
treasuryGuardian	address	32	0	20
treasuryAddress	address	33	0	20
treasuryPercent	uint256	34	0	32
venusContributorSp	mapping(address	35	0	32

Name	Type	Slot	Offset	Bytes
eeds	=> uint256)			
lastContributorBlock	mapping(address => uint256)	36	0	32
liquidatorContract	address	37	0	20
comptrollerLens	contract ComptrollerLensInterface	38	0	20
supplyCaps	mapping(address => uint256)	39	0	32
accessControl	address	40	0	20
_actionPaused	mapping(address => mapping(uint256 => bool))	41	0	32
venusBorrowSpeeds	mapping(address => uint256)	42	0	32
venusSupplySpeeds	mapping(address => uint256)	43	0	32
approvedDelegates	mapping(address => mapping(address => bool))	44	0	32

Storage of `Diamond.sol` and its Facets (Replacement for `Comptroller.sol`)

Name	Type	Slot	Offset	Bytes
admin	address	0	0	20
pendingAdmin	address	1	0	20
comptrollerImplementation	address	2	0	20
pendingComptrollerImplementation	address	3	0	20
oracle	contract PriceOracle	4	0	20
closeFactorMantissa	uint256	5	0	32

Name	Type	Slot	Offset	Bytes
liquidationIncentiveMantissa	uint256	6	0	32
maxAssets	uint256	7	0	32
accountAssets	mapping(address => contract VToken[])	8	0	32
markets	mapping(address => struct ComptrollerV1Storage.Market)	9	0	32
pauseGuardian	address	10	0	20
_mintGuardianPaused	bool	10	20	1
_borrowGuardianPaused	bool	10	21	1
transferGuardianPaused	bool	10	22	1
seizeGuardianPaused	bool	10	23	1
mintGuardianPaused	mapping(address => bool)	11	0	32
borrowGuardianPaused	mapping(address => bool)	12	0	32
allMarkets	contract VToken[]	13	0	32
venusRate	uint256	14	0	32
venusSpeeds	mapping(address => uint256)	15	0	32
venusSupplyState	mapping(address => struct ComptrollerV1Storage.VenusMarketState)	16	0	32
venusBorrowState	mapping(address => struct ComptrollerV1Stora	17	0	32

Name	Type	Slot	Offset	Bytes
	ge.VenusMarketState)			
venusSupplierIndex	mapping(address => mapping(address => uint256))	18	0	32
venusBorrowerIndex	mapping(address => mapping(address => uint256))	19	0	32
venusAccrued	mapping(address => uint256)	20	0	32
vaiController	contract VAIControllerInterface	21	0	20
mintedVAIs	mapping(address => uint256)	22	0	32
vaiMintRate	uint256	23	0	32
mintVAIGuardianPaused	bool	24	0	1
repayVAIGuardianPaused	bool	24	1	1
protocolPaused	bool	24	2	1
venusVAIRate	uint256	25	0	32
venusVAIVaultRate	uint256	26	0	32
vaiVaultAddress	address	27	0	20
releaseStartBlock	uint256	28	0	32
minReleaseAmount	uint256	29	0	32
borrowCapGuardian	address	30	0	20
borrowCaps	mapping(address => uint256)	31	0	32
treasuryGuardian	address	32	0	20

Name	Type	Slot	Offset	Bytes
treasuryAddress	address	33	0	20
treasuryPercent	uint256	34	0	32
venusContributorSpeeds	mapping(address => uint256)	35	0	32
lastContributorBlock	mapping(address => uint256)	36	0	32
liquidatorContract	address	37	0	20
comptrollerLens	contract ComptrollerLensInterface	38	0	20
supplyCaps	mapping(address => uint256)	39	0	32
accessControl	address	40	0	20
_actionPaused	mapping(address => mapping(uint256 => bool))	41	0	32
venusBorrowSpeeds	mapping(address => uint256)	42	0	32
venusSupplySpeeds	mapping(address => uint256)	43	0	32
approvedDelegates	mapping(address => mapping(address => bool))	44	0	32
selectorToFacetAndPosition	mapping(bytes4 => struct ComptrollerV12Storage.FacetAddressAndPosition)	45	0	32
facetFunctionSelectors	mapping(address => struct ComptrollerV12Storage.FacetFunctionSelectors)	46	0	32
facetAddresses	address[]	47	0	32

FINDINGS

VENUS - DIAMOND COMPTROLLER (SUBSCRIPTION AUDIT 5)

10	0	2	0	1	7
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Venus - Diamond Comptroller (Subscription Audit 5). Through this audit, we have uncovered 10 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
DCV-02	**Centralization Related Risks**	**Centralization**	**Major**	**Mitigated**
DDC-03	**Centralized Control Of Contract Upgrade**	**Centralization**	**Major**	**Mitigated**
SFD-01	Missing Check May Cause Functions With `releaseToVault()` To Get Stuck	Logical Issue	Minor	Acknowledged
CVP-01	Potential Difficulty In Accommodating Upgrades	Design Issue, Volatile Code	Informational	Resolved
DCV-01	Unprotected Direct Updates To State Of `Diamond.sol` And Facets Through Function `updateDelegate()`	Design Issue, Access Control	Informational	Resolved
DDC-01	Immutable Functions Of `Diamond.sol`	Coding Style	Informational	Resolved
DDC-02	No Initializing Logic When Functions Are Added, Removed, Replaced	Logical Issue	Informational	Resolved
DDC-04	`Diamond.sol` Does Not Implement The `DiamondLoupe` Interface According To EIP-2535 Specification	Coding Style	Informational	Resolved
DDC-05	Potential For Overflow	Coding Issue	Informational	Resolved

ID	Title	Category	Severity	Status
VAI-01	Unnecessary Remnant Casting	Code Optimization	Informational	Resolved

DCV-02 | CENTRALIZATION RELATED RISKS

Category	Severity	Location	Status
Centralization	Major	contracts/Comptroller/Diamond/Diamond.sol (base): 24~25; contracts/Comptroller/Diamond/facets/MarketFacet.sol (base): 167~168; contracts/Comptroller/Diamond/facets/RewardFacet.sol (base): 108~109; contracts/Comptroller/Diamond/facets/SetterFacet.sol (base): 74~75, 97~98, 114~115, 133~134, 171~172, 187~188, 200~201, 221~222, 241~242, 258~259, 272~273, 301~302, 313~314, 342~343, 374~375, 388~389, 404~405	Mitigated

Description

In the contract `MarketFacet` the role set for `"_supportMarkt(address)"` has authority over the functions listed below

- `_supportMarket()`

Any compromise to this account may allow a hacker to take advantage of this authority and add a malicious token address they control as a market allowing them to steal other assets.

In the contract `RewardFacet` the role `admin` or `comptrollerImplementation` has authority over the functions listed below

- `_grantXVS()`

Any compromise to either account may allow a hacker to take advantage of this authority and send any address the full balance of `XVS` held by the Comptroller.

In the contract `SetterFacet` the role `admin` has authority over the functions listed below

- `_setPriceOracle()`
- `_setCloseFactor()`
- `_setAccessControl()`
- `_setLiquidatorContract()`
- `_setPauseGuardian()`
- `_setVAIController()`
- `_setVAIMintRate()`
- `_setTreasuryData()`
- `_setComptrollerLens()`
- `_setVenusVAIVaultRate()`

- `_setVAIVaultInfo()`

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

- change the `oracle` to a malicious one.
- update the `closeFactorMantissa` to any value, even outside the maximum and minimum specified range.
- change the `accessControl` contract so that other privileged functions can be exploited.
- change the `liquidatorContract` to one they control.
- change the pause guardian in order to block functionality the guardian controls.
- update the `vaiController` which is an address that can update the `mintedVAIs` mapping for any address with any amount.
- adjust the `vaiMintRate` used in determining the amount of mintable VAI in the `VAIController` contract.
- change the `treasuryAddress` to one they control, and adjust the `treasuryPercent` to the maximum value.
- change the `comptrollerLens`
- update information about the VAI Vault such as its rate of issuance, the `vaiVaultAddress`, the start of its release, and the minimum amount needed for release to the vault.

The following functions have their own assigned privileged account based on their function signature:

- `_setCollateralFactor()`
- `_setLiquidationIncentive()`
- `_setMarketBorrowCaps()`
- `_setMarketSupplyCaps()`
- `_setActionsPaused()`

Any compromise to the one of the accounts with the associated privilege may allow a hacker to take advantage of this authority and

- change the collateral factor or liquidation incentive to values that allow for manipulation of the protocol.
- pause actions to prevent users' ability to, for example, exit markets, or to prevent a position from being liquidated.
- adjust the market borrowCaps and supplyCaps so that, for example, a larger borrow is possible.

The role `treasuryGuardian` has authority over `_setTreasuryData()`. Any compromise to the `treasuryGuardian` may allow a hacker to take advantage of this authority and change the `treasuryAddress` to one they control, and adjust the `treasuryPercent` to the maximum value.

In the contract `Diamond.sol` the role `admin` has authority over the functions listed below

- `diamondCut()`

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

- remove function signature assignments from the storage of the contract, preventing use of the functions from the `Unitroller`
- incorrectly set up the association between a function signature and a facet address
- use a malicious facet address for implementation of a function in the `Unitroller`

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 recommend carefully managing 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., multi-signature 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 of privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key being 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 of 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 08/02/2023]`: The admin of the Unitroller contract is 0x939bd8d64c0a9583a7dcea9933f7b21697ab6396, that 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, only the community, via a VIP will be able to execute the mentioned protected functions restricted for the admin.

We'll use the AccessControlManager (ACM) deployed at
https://bscscan.com/address/0x4788629abc6cfca10f9f969efdeaa1cf70c23555

In this ACM, only 0x939bd8d64c0a9583a7dcea9933f7b21697ab6396 (Normal) has the DEFAULT_ADMIN_ROLE. And this contract is a Timelock contract used during the Venus Improvement Proposals. The idea is not to add new authorized accounts for the mentioned functions. The address 0x939bd8d64c0a9583a7dcea9933f7b21697ab6396 is already granted to execute every mentioned function protected by the ACM.

DDC-03 | CENTRALIZED CONTROL OF CONTRACT UPGRADE

Category	Severity	Location	Status
Centralization	**Major**	**contracts/Comptroller/Diamond/Diamond.sol (base): 15~18**	**Mitigated**

Description

`Diamond.sol` is an upgradeable contract, the `admin` can upgrade the contract without the community's commitment. If an attacker compromises the account, he can change the implementation of the contract and drain tokens from the contract.

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 08/02/2023]` : Diamond.sol will be the implementation contract of the Unitroller contract, already deployed at 0xfd36e2c2a6789db23113685031d7f16329158384. The admin of Unitroller contract is 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, every upgrade will be done only via a Normal VIP, involving the community in the process.

SFD-01 MISSING CHECK MAY CAUSE FUNCTIONS WITH `releaseToVault()` TO GET STUCK

Category	Severity	Location	Status
Logical Issue	Minor	contracts/Comptroller/Diamond/facets/SetterFacet.sol (base): 390~392, 407~409, 411~412	Acknowledged

Description

Functions `_setVenusVAIVaultRate()` and `_setVAIVaultInfo()` were updated to include the following logic:

```
if (vaiVaultAddress != address(0)) {
            releaseToVault();
        }
```

This check is also performed in functions `distributeSupplierVenus()` and `distributeBorrowerVenus()`.

In function `_setVAIVaultInfo()`, the `vaiVaultAddress` can be updated to any address. Function `releaseToVault()` however uses address `vaiVaultAddress` in interface `IVAIVault` to call function `updatePendingRewards()`. If the address `vaiVaultAddress` is set to an externally owned account, or a contract which does not include this function, then the call will revert, which keeps any calls to the functions that make the call to `releaseToVault()` revert. Since `_setVAIVaultInfo()` is the function used to update the address, this would prevent the function from being called to correct the issue.

The severity is set to minor since the contract logic can be updated to change the logic in the scenario where this occurs.

Recommendation

We recommend adding extra checks to `_setVAIVault()` to ensure the `vaiVaultAddress` is updated to a contract which implements the function `updatePendingRewards()`.

Alleviation

`[CertiK]`: The team acknowledges the finding and opts not to change the current version. They note that addresses are only updated by governance, making the possibility of error negligible.

CVP-01 | POTENTIAL DIFFICULTY IN ACCOMMODATING UPGRADES

Category	Severity	Location	Status
Design Issue, Volatile Code	Informational	contracts/Comptroller/ComptrollerStorage.sol (base): 238~239; contracts/Comptroller/Diamond/Diamond.sol (base): 8~9; contracts/Comptroller/Diamond/facets/FacetBase.sol (base): 10~11; contracts/Comptroller/Diamond/facets/MarketFacet.sol (base): 9~10; contracts/Comptroller/Diamond/facets/PolicyFacet.sol (base): 10~11; contracts/Comptroller/Diamond/facets/RewardFacet.sol (base): 9~10; contracts/Comptroller/Diamond/facets/SetterFacet.sol (base): 11~12; contracts/Comptroller/Diamond/facets/XVSRewardsHelper.sol (base): 8~9	Resolved

Description

Contract `Diamond.sol` does not meet all of the required specifications of EIP-2535. One of the main distinctions is that EIP-2535 specifies that a `Diamond` contract should be the proxy contract, holding its own state, while the facets it associates with are its stateless logic contracts.

Instead, the intention of the protocol is to keep the already deployed `Unitroller` contract as the stateful proxy, and use the new `Diamond` contract as a logic contract that outsources the main Comptroller logic to various facets. Additionally, each facet and the `Diamond` contract inherit the eternal storage pattern held in the `Unitroller` in order to facilitate the retention of the old logic patterns implemented in the Comptroller logic contract that is currently being used.

This difference in set up could potentially cause difficulty in future upgrades.

1. Any time an upgrade is made to the logic of any facet that introduces a new state variable, every facet may need to be updated in order to keep a consistent reference to the eternal storage of the `Unitroller`. It is noted that the update of each facet to reflect all current state variables in the `Unitroller` would not necessarily need to be immediate if the facet does not use the new state variable in its current logic. However, if each facet does not maintain a consistent reference to the `Unitroller` storage, this could cause issues with the storage variables referenced in updated logic. An effort to maintain a consistent storage reference throughout all facets would mean that each facet contract address would have to be updated every time there is any update to storage.
2. The difficulty presented in necessitating the replacement/removal of every old facet during upgrade could lead to vulnerabilities if the update is not handled atomically. Adding, replacing, or removing one function or facet at a time could allow for unexpected points of failure in the code, or unintended entry points.
3. Additionally the `Diamond.sol` contract only currently inherits `ComptrollerV12Storage`. If new state variables are added, it appears that it is necessary to implement a new `Diamond` contract for the implementation used in the `Unitroller`.

Recommendation

We recommend sharing the intended plan for future upgrades in handling the potential issues above.

Alleviation

`[Venus]` : "Issue acknowledged. I won't make any changes for the current version. We will need to upgrade only the facets for which the new state has been added with the updated storage layout. Every update will be done via VIP. So, if we need to do several changes, all of them will be included in the same VIP and executed in the same transaction."

DCV-01 UNPROTECTED DIRECT UPDATES TO STATE OF `Diamond.sol` AND FACETS THROUGH FUNCTION `updateDelegate()`

Category	Severity	Location	Status
Design Issue, Access Control	Informational	contracts/Comptroller/Diamond/Diamond.sol (base): 215~216; contracts/Comptroller/Diamond/facets/MarketFacet.sol (base): 198~199	Resolved

Description

EIP-2535 specifies that the `Diamond` contract should act as stateful proxy, and that its included facets should be stateless.

However, in the upgrade to the new logic pattern employed by this project, the `Diamond` contract also acts as a logic contract, keeping the originally deployed `Unitroller` as the stateful proxy.

Even so, both the `Diamond` and the included facets inherit the Comptroller's eternal storage, and, as a result, are both technically stateful, where some storage variables can be updated directly within each contract, outside of the use of the `Unitroller`.

It is noted that addresses such as `admin` and `accessControl` cannot be updated directly within these contracts. Moreover, many user-facing endpoints contain a check to `ensureListed()` which will always return false. Thus, in most cases, the state cannot be updated.

The only function that was found to successfully update the state of these contracts directly was function `updateDelegate()` which can be called directly in the `Diamond` or `MarketFacet` contract to update the state of either contract.

A malicious user may update the state of either the `Diamond` or `MarketFacet` directly in an attempt to feed other users false information about the protocol.

Recommendation

We recommend considering the inclusion of logic in the cited function which prevents the direct update of the state of the relevant contracts.

Acknowledgement of the state is enough to resolve the finding in this case, if the recommended change is not desirable for the protocol.

Alleviation

`[CertiK]`: The team acknowledges the information presented in the finding.

DDC-01 | IMMUTABLE FUNCTIONS OF `Diamond.sol`

Category	Severity	Location	Status
Coding Style	Informational	contracts/Comptroller/Diamond/Diamond.sol (base): 15~16, 24~25, 34~35, 43~44, 51~52, 60~61	Resolved

Description

EIP-2535 contains specifications for how *immutable* functions should be handled. Namely, "any attempt to replace or remove an immutable function must revert."

Reference: https://eips.ethereum.org/EIPS/eip-2535#addingreplacingremoving-functions

There are no requirements within the specification for which functions are required to be made immutable, and a Diamond contract may not hold any immutable functions. Please specify whether any of the functions within `Diamond.sol` are intended to be considered as immutable for the project. If so, we recommend including logic within the logical path of `diamondCut()` to ensure that these functions cannot be replaced or removed, as the specification requires.

Recommendation

We recommend including logic within the logical path of `diamondCut()` to ensure that any functions intended to be immutable cannot be replaced or removed, as the specification requires.

If no functions within the `Diamond.sol` implementation and its facets are intended to be immutable and this logic is not necessary, please provide a statement verifying this.

Alleviation

`[CertiK]`: The team states they do not currently have a need for immutable functions, so they do not currently need to include checks for immutable functions.

DDC-02 NO INITIALIZING LOGIC WHEN FUNCTIONS ARE ADDED, REMOVED, REPLACED

Category	Severity	Location	Status
Logical Issue	Informational	contracts/Comptroller/Diamond/Diamond.sol (base): 24~25	Resolved

Description

It is noted that there is no logic to handle the potential need to initialize any new states that may be introduced through upgrades to Comptroller when new functions are added, removed, or replaced. Please state the plan for initialization when updates necessitate this.

Recommendation

We recommend providing the plan for initialization, in the case where it is necessary.

Alleviation

`[Venus]` : "We would invoke the needed setter functions, from the VIP, if there would be a need to initialize the state."

DDC-04 `Diamond.sol` DOES NOT IMPLEMENT THE `DiamondLoupe` INTERFACE ACCORDING TO EIP-2535 SPECIFICATION

Category	Severity	Location	Status
Coding Style	Informational	contracts/Comptroller/Diamond/Diamond.sol (base): 34~35, 51~52, 60~61	Resolved

Description

One requirement of the EIP-2535 specification on Diamond proxies is that the Diamond contract in use must implement the DiamondLoupe interface, either through inheritance or inclusion in a facet of the Diamond.

References:

https://eips.ethereum.org/EIPS/eip-2535#implementation-points

https://eips.ethereum.org/EIPS/eip-2535#a-note-on-implementing-interfaces

The `Diamond.sol` file of the project implements some view functions which are similar to those listed in the outlined `IDiamondLoupe` interface, but these functions do not have the same name:

- `getFacetFunctionSelectors()` in `Diamond.sol` performs the same actions as what is specified for the function `facetFunctionSelectors()` in the `IDiamondLoupe` interface;
- `getAllFacetAddresses()` in `Diamond.sol` performs the same actions as what is specified for the function `facetAddresses()` in the `IDiamondLoupe` interface;
- `getFacetAddressAndPosition()` in `Diamond.sol` performs some of the same actions as what is specified for the function `facetAddress()` in the `IDiamondLoupe` interface;

Additionally, there is one function of the `IDiamondLoupe` interface which is not implemented by `Diamond.sol`, that is function `facets()`, which is supposed to return an array of `Facet` structs for all facet addresses used within `Diamond.sol`. The `Facet` struct is not currently used within the project.

Recommendation

To be in accordance with EIP-2535 as much as possible, we recommend taking the following steps:

1. Rename function `getFacetFunctionSelectors()` to `facetFunctionSelectors()` in `Diamond.sol`
2. Rename function `getAllFacetAddresses()` to `facetAddresses()` in `Diamond.sol`
3. Refactor `getFacetAddressAndPosition()` to follow the expected return values of function `facetAddress()` in the `IDiamondLoupe` interface, and rename the function accordingly. Alternatively, create a `facetAddress()` function that implements the specification of in `IDiamondLoupe`;

4. Consider adding in and updating an array of `Facet` struct inputs to return information for according to function `facet()` in the `IDiamondLoupe` interface.

Alleviation

`[CertiK]`: The team made changes resolving the finding in commit 7417d8f4b17eb156dd44a8b4d8eb6dbf3e6e4015.

DDC-05 | POTENTIAL FOR OVERFLOW

Category	Severity	Location	Status
Coding Issue	Informational	contracts/Comptroller/Diamond/Diamond.sol (base): 94~95, 96~97, 116~117, 118~119, 164~165, 178~179	Resolved

Description

Potential for overflow with `selectorPosition` is technically possible if more than $2^{96} - 1$ function selectors are added to the `Diamond` contract for a given facet address. It is recognized that the occurrence is unlikely without malicious takeover of the privileged `admin` account since this number of functions exceeds what is practical for any given deployed contract.

If the value of `selectorPosition` for a given `_facetAddress` exceeds $2^{96} - 1$ then

- overflow to a value of 0 may cause `addFacet()` to be called on a `_facetAddress` which is already included in `facetAddresses` array
- the selector's recorded `functionSelectorPosition` in mapping `selectorToFacetAndPosition` may not accurately record its actual position in the `functionSelectors` array of the `_facetAddress` in `facetFunctionSelectors`.

Recommendation

We recommend keeping this information in consideration during updates of the `Diamond.sol` contract.

The finding will be set to resolved upon acknowledgement of the above information.

Alleviation

`[CertiK]`: The team acknowledges the above information.

VAI-01 | UNNECESSARY REMNANT CASTING

Category	Severity	Location	Status
Code Optimization	Informational	contracts/Tokens/VAI/VAIController.sol (base): 453~454	Resolved

Description

Function `getMintableVAI()` of contract `VAIController` uses the following structure for calling `markets()`:

```
            (, uint collateralFactorMantissa) =
ComptrollerInterface(address(comptroller)).markets(
```

This casting structure is unnecessary since `comptroller` is already of type `ComptrollerInterface` and can be use directly to call `markets()`.

Recommendation

We recommend streamlining the codebase by replacing `ComptrollerInterface(address(comptroller))` with `comptroller`.

Alleviation

`[CertiK]`: The team made changes resolving the finding in commit 53a08eb7b0d2ad567842660d76a5a7dc9a0d8a34.

APPENDIX | VENUS - DIAMOND COMPTROLLER (SUBSCRIPTION AUDIT 5)

Finding Categories

Categories	Description
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.
Access Control	Access Control findings are about security vulnerabilities that make protected assets unsafe.
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.
Design Issue	Design Issue findings indicate general issues at the design level beyond program logic that are not covered by other finding categories.

Checksum Calculation Method

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

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

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.