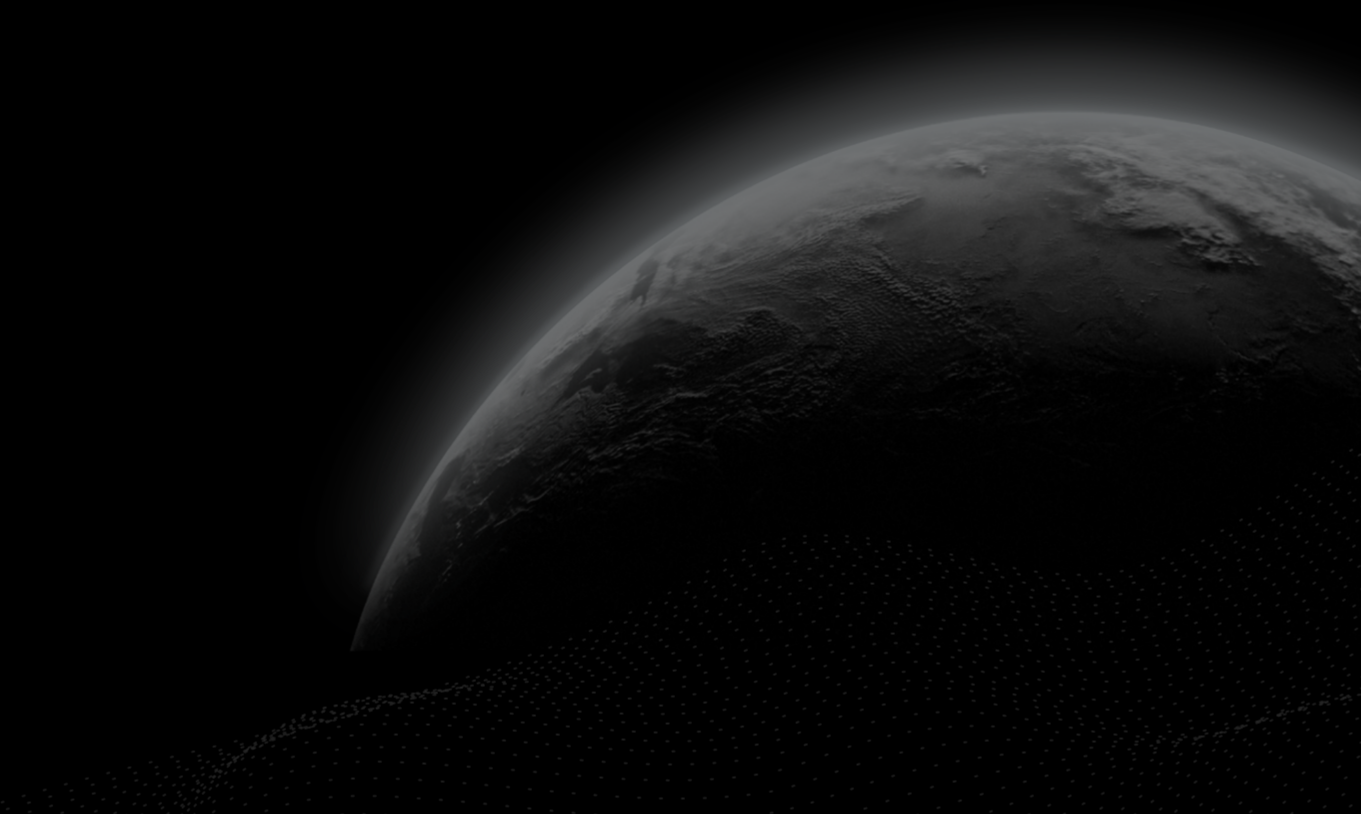
Security Assessment

Venus - Oracle

CertiK Assessed on May 22nd, 2023



CertiK Assessed on May 22nd, 2023

Venus - Oracle

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 05/22/2023	N/A

CODEBASE

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

...View All

COMMITS

base: 5386ce8732a397cbe2e8317cc051e306f4eacff8

update1: 849ffd563e60a45780eae4d1126983e7b32d9ed6

update2: 8fa1becb9b1c512e0b68d73dff09ee4aa172c882

...View All

Vulnerability Summary

11	9	2	0	0	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.
🟨 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.
▨ 1	Minor	1 Resolved	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
🟦 6	Informational	6 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 - ORACLE

CODEBASE | VENUS - ORACLE

Repository

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

Commit

base: 5386ce8732a397cbe2e8317cc051e306f4eacff8

update1: 849ffd563e60a45780eae4d1126983e7b32d9ed6

update2: 8fa1becb9b1c512e0b68d73dff09ee4aa172c882

AUDIT SCOPE | VENUS - ORACLE

13 files audited ● 1 file with Acknowledged findings ● 5 files with Mitigated findings ● 7 files without findings

ID	Repo	Commit	File	SHA256 Checksum
● COV	VenusProtocol/oracle	5386ce8	contracts/oracles/ChainlinkOracle.sol	8819d69ff8aa6b31299f593a754f384dc8693e432248a9ffccd75e3792ebdc8d
● ROV	VenusProtocol/oracle	5386ce8	contracts/ResilientOracle.sol	c91feb9b9f0b7a7a4d6bfcdbbd8e83c32f49ea7596a2d7858fe4e6319715ea51
● BOV	VenusProtocol/oracle	5386ce8	contracts/oracles/BinanceOracle.sol	eacf7f437553380e8d8681179ad97e7850c1e4862bc9fa7bf5c24734ba47f69a
● BVV	VenusProtocol/oracle	5386ce8	contracts/oracles/BoundValidator.sol	e4ab515f8e83008eccec8cb5b7d31837f5746c707e2366c778640875bda2c99e
● POV	VenusProtocol/oracle	5386ce8	contracts/oracles/PythOracle.sol	54ff14fbe54f28c344bbc9306db16b93bce5dae4fd2eb0d3a363847475122af7
● TOV	VenusProtocol/oracle	5386ce8	contracts/oracles/TwapOracle.sol	d9497e3ae44a1d584ad8b3e02432acfe570cfff7f18d0951582219df8249b4fc
● FRI	VenusProtocol/oracle	5386ce8	contracts/interfaces/FeedRegistryInterface.sol	cad4841a41bb5d2016f025e0b9be401e980d7e7dd6a564a9c829ac092aab2574
● OIV	VenusProtocol/oracle	5386ce8	contracts/interfaces/OracleInterface.sol	2cdabe0f3287911fde6837d78568780a5c3619d9a0ce6e654e3c935af4e79915
● PRI	VenusProtocol/oracle	5386ce8	contracts/interfaces/PublicResolverInterface.sol	6a5fc13054cd05b787b161993f62275e1661fed2476fa4240bef7a515b3eaa0a
● PIV	VenusProtocol/oracle	5386ce8	contracts/interfaces/PythInterface.sol	d1789f5c3ab73b70077bf36c498e19efc18de19386ecdf94afc3adf283dfb1ab
● SID	VenusProtocol/oracle	5386ce8	contracts/interfaces/SIDRegistryInterface.sol	8e900f5ff77d6d6e015751408b3a365dc8850046d1a8efcd707aca8300cb16d4
● VBI	VenusProtocol/oracle	5386ce8	contracts/interfaces/VBep20Interface.sol	8e33f4d371da4e2ae4a52537fd73e26d70c10c41e1298399b386daf32fa02546

ID	Repo	Commit	File	SHA256 Checksum
PLV	VenusProtocol/oracle	5386ce8	contracts/libraries/PancakeLibrary.sol	cd85bbbfb29f528174da8ab5b53129c89c9faa37f47e8ccca826273d6cda1389

APPROACH & METHODS | VENUS - ORACLE

This report has been prepared for Venus to discover issues and vulnerabilities in the source code of the Venus - Oracle 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 - ORACLE

BoundValidator

This contract is designed to set bounds on the ratio of two reported prices for an asset. It is also then used to check that the ratio of these prices lie within the specified bounds.

BinanceOracle

This contract is designed to interact safely with the Binance Oracle. It checks that the price returned is not too old, chooses the correct asset for the underlying of vTokens or VAI, and returns the normalized price.

ChainlinkOracle

This contract is designed to interact safely with the Chainlink Oracle. It checks that the price returned is not too old, chooses the correct asset for the underlying of vTokens or VAI, and returns the normalized price. In addition, it adds functionality to add direct prices, which if a direct price is set for an asset, then it will be returned instead of the price provided by the Chainlink oracle. This allows the price for assets to be set by anyone with privilege to these functions (see centralization risk finding), which should only be used in extenuating conditions such as price feed failure.

PythOracle

This contract is designed to interact safely with the Pyth Oracle. It checks that the price returned is not too old, chooses the correct asset for the underlying of vTokens or VAI, and returns the normalized price.

TwapOracle

This contract is designed to interact with PancakeSwap to fetch the cumulative prices and update them to the current block.timestamp to calculate the time weighted average price (TWAP).

ResilientOracle

This contract is designed to interact with three separate oracles for each asset. One oracle will be the main oracle, which will be the first choice for price. The second oracle will be the pivot oracle, whose price is used for comparison. The third oracle is the fallback oracle, whose price will be used in certain cases.

First, the main oracles price is compared against the pivot oracles price. If the pivot oracle is not enabled then the main oracle's price will be returned provided the main oracle is enabled and not the zero address. If the pivot oracle is enabled, then the main oracles price will be checked to be within a specified range of the pivot oracles price. If it is within this range, then the main oracle price is returned. If it is not in the range it will move on to the fallback oracle.

If the validation of the main oracle price vs. the pivot oracle price fails, then it will use the fallback oracle provided it is enabled and not the zero address. In this case, it will check if the fallback price is within the specified range of the pivot oracle price. If it is within this range, then the fallback oracle price is returned. If it is not in the range it will attempt to validate the main oracle price with the fallback oracle price.

Lastly, if both validations for the main oracle and fallback oracle vs. the pivot oracle fail, then the main oracle price and fallback oracle price are compared to see if they lie in the specified range. If they do, then the main oracle price is returned. Otherwise, the call will revert.

DEPENDENCIES | VENUS - ORACLE

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:

- Chainlink Oracle
- Binance Oracle
- PythOracle
- AMM's Such As PancakeSwap

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. In addition, upgrades of third parties can possibly create severe impacts, such as increasing fees of third parties, migrating to new LP pools, etc.

Recommendations

We recommend constantly monitoring the third parties involved to mitigate any side effects that may occur when unexpected changes are introduced.

FINDINGS | VENUS - ORACLE

11	0	2	2	1	6
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Venus - Oracle. Through this audit, we have uncovered 11 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
VPB-03	**Centralization Related Risks**	**Centralization / Privilege**	**Major**	● **Mitigated**
VPB-04	**Centralized Control Of Contract Upgrade**	**Centralization / Privilege**	**Major**	● **Mitigated**
BOV-01	`BinanceOracle` Does Not Properly Implement `OracleInterface`	Logical Issue	Medium	● Resolved
COV-01	Chainlink Can Return Negative Price That Will Not Be Reverted	Logical Issue	Medium	● Resolved
ROV-02	Missing Zero Address Validation	Inconsistency	Minor	● Resolved
COV-02	Incorrect Emit Event	Logical Issue	Informational	● Resolved
POV-02	Unnecessary Casting	Inconsistency	Informational	● Resolved
ROV-01	`fallbackPrice` Is Tested Against `mainPrice`	Logical Issue	Informational	● Resolved
TOV-01	Missing Checks For Feeds And Pools	Logical Issue	Informational	● Resolved
VPB-01	Typos And Inconsistencies	Inconsistency	Informational	● Resolved
VPB-05	Access Control Convention	Logical Issue	Informational	● Resolved

VPB-03 | CENTRALIZATION RELATED RISKS

Category	Severity	Location	Status
Centralization / Privilege	● **Major**	**contracts/ResilientOracle.sol (base): 98, 107, 118, 143, 163, 267; contracts/oracles/BinanceOracle.sol (base): 45; contracts/oracles/BoundValidator.sol (base): 59, 86; contracts/oracles/ChainlinkOracle.sol (base): 75, 90, 103, 131; contracts/oracles/PythOracle.sol (base): 76, 94, 120; contracts/oracles/TwapOracle.sol (base): 104, 153**	● **Mitigated**

Description

In the contract `BinanceOracle` , the `DEFAULT_ADMIN_ROLE` can grant access to the following functions:

- `setMaxStalePeriod()`

Any compromise to the `DEFAULT_ADMIN_ROLE` or access to the functions may allow a hacker to do the following:

- set the `maxStalePeriod` to any nonzero value. If they set the value to be very large, then this allows old prices to be valid. If the value is set to be very small, then reasonably recent prices will be considered invalid.

In the contract `BoundValidator` , the `DEFAULT_ADMIN_ROLE` can grant access to the following functions:

- `setValidateConfigs()`
- `setValidateConfig()`

Any compromise to the `DEFAULT_ADMIN_ROLE` or access to the functions may allow a hacker to do the following:

- set the upper and lower bound validation ratios for an asset. In particular this allows them to set the ratio to be a very small range, in which case most time the price will not be validated. Or they can set the ratio to a large range, allowing prices to be validated when they are not reasonably close to one another.

In the contract `ChainlinkOracle` , the `DEFAULT_ADMIN_ROLE` can grant access to the following functions:

- `setUnderlyingPrice()`
- `setDirectPrice()`
- `setTokenConfigs()`
- `setTokenConfig()`

Any compromise to the `DEFAULT_ADMIN_ROLE` or access to the functions may allow a hacker to do the following:

- change the forced prices for assets or the underlying assets of `vToken`. If only this oracle is used as the main oracle, this would allow the hacker to set the exact price they want for an asset. If it is used as the pivot, then the value can be set to always validate the fallback or main oracle, even if the oracle is compromised and returns unreasonable prices. If it is used as the fallback, it can be used to get the best price that the pivot would validate or to validate the main oracles price, even if it is unreasonable. If this is the only oracle used, then this allows a hacker to set the price they want for an asset.
- set the feed address and `maxStalePeriod` for an asset. In particular a hacker could set the feed address of the asset to a feed that is not for the asset and USD and use the incorrect price to exploit funds from the protocol. The hacker can also set the `maxStalePeriod` to a small value, so that reasonably recent prices are invalid, or to a large value so that old prices may be used.

In the contract `PythOracle`, the `DEFAULT_ADMIN_ROLE` can grant access to the following functions:

- `setUnderlyingPythOracle()`
- `setTokenConfigs()`
- `setTokenConfig()`

Any compromise to the `DEFAULT_ADMIN_ROLE` or access to the functions may allow a hacker to do the following:

- set the `underlyingPythOracle` to an address of a malicious contract that will return incorrect prices that can be used to exploit the protocol.
- set the `pythId` and `maxStalePeriod` for an asset. In particular a hacker could set the `pythId` of the asset to a feed that is not for the asset and USD and use the incorrect price to exploit funds from the protocol. The hacker can also set the `maxStalePeriod` to a small value, so that reasonably recent prices are invalid, or to a large value so that old prices may be used.

In the contract `TwapOracle`, the `DEFAULT_ADMIN_ROLE` can grant access to the following functions:

- `setTokenConfigs()`
- `setTokenConfig()`

Any compromise to the `DEFAULT_ADMIN_ROLE` or access to the functions may allow a hacker to do the following:

- set the `baseUnit`, `pancakePool`, `isBnbBased`, `isReversedPool`, and `anchorPeriod` for any asset. A hacker can change these values to manipulate the price that is given for the asset to exploit funds from the protocol.

In the contract `ResilientOracle`, the `DEFAULT_ADMIN_ROLE` can grant access to the following functions:

- `pause()`
- `unpause()`

- `setOracle()`
- `enableOracle()`
- `setTokenConfigs()`
- `setTokenConfig()`

Any compromise to the `DEFAULT_ADMIN_ROLE` or access to the functions may allow a hacker to do the following:

- pause the oracle, so that any call to `getUnderlyingPrice()` will revert. This can allow a hacker to perform a denial of service attack.
- unpause the oracle, allowing `getUnderlyingPrice()` to be called. This can allow the hacker to exploit the protocol if it was paused due to a bug.
- set the main, pivot, or fallback oracles for an asset. A hacker could change these addresses to malicious contracts that will return incorrect prices allowing the hacker to exploit funds from the protocol.
- set if the main, pivot, or fallback oracles are enabled for an asset. If a hacker has compromised the main oracle, they can disable the pivot so that the main price will be used and the hacker can use the incorrect price to exploit funds from the protocol. They can also perform a denial of service by disabling the oracles.

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 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]` : 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 to grant 0x939bd8d64c0a9583a7dcea9933f7b21697ab6396 to execute the mentioned functions. Moreover, we'll allow [a] (Fast-track) and [b] (Critical) also to execute pause() and unpause() functions in the ResilientOracle. These are the Timelock contracts to execute VIP's with a shorter delay.

Specifically, the current config for the three Timelock contracts are:

- normal: 24 hours voting + 48 hours delay
- fast-track: 24 hours voting + 6 hours delay
- critical: 6 hours voting + 1 hour delay

Regarding the role, specifically, the sequence in the ACM was:

- In [1] the ACM was created, and the address 0x55a9f5374af30e3045fb491f1da3c2e8a74d168d had the DEFAULT_ADMIN_ROLE.
- In [2], 0x55a9f5374af30e3045fb491f1da3c2e8a74d168d gave the DEFAULT_ADMIN_ROLE to 0x939bd8d64c0a9583a7dcea9933f7b21697ab6396.
- In [3] 0x55a9f5374af30e3045fb491f1da3c2e8a74d168d renounced to the DEFAULT_ADMIN_ROLE.

Therefore, we consider this setup safe enough and don't plan to do any other changes.

[a] https://bscscan.com/address/0x555ba73dB1b006F3f2C7dB7126d6e4343aDBce02

[b] https://bscscan.com/address/0x213c446ec11e45b15a6E29C1C1b402B8897f606d

[1] https://bscscan.com/tx/0x3eb2ef9b54b1ec3873e07fc9994d32de6fe6c9bc9277c17619c6fa6701340ae0

[2] https://bscscan.com/tx/0x66b32b0d8918b43e43e2b6104927273f012b81ad8ee30d1284c6067aa761b687

[3] https://bscscan.com/tx/0x2a4b3b21f5acd9fb73c9fa740d9a8a123780bdb01ec712baac639576df33d7d4

VPB-04 | CENTRALIZED CONTROL OF CONTRACT UPGRADE

Category	Severity	Location	Status
Centralization / Privilege	● **Major**	**contracts/ResilientOracle.sol (base): 247; contracts/oracles/BinanceOracle.sol (base): 58; contracts/oracles/BoundValidator.sol (base): 72; contracts/oracles/ChainlinkOracle.sol (base): 115; contracts/oracles/PythOracle.sol (base): 104; contracts/oracles/TwapOracle.sol (base): 134**	● **Mitigated**

Description

`BinanceOracle` , `BoundValidator` , `ChainlinkOracle` , `PythOracle` , `TwapOracle` , and `ResilientOracle` are upgradeable contracts. The owner 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 as well as change the logic of the contract to return incorrect prices.

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 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]` : The ownership of these contracts will be transferred to 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, these contracts will be upgraded only via a Normal VIP, involving the community in the process.

BOV-01 | `BinanceOracle` DOES NOT PROPERLY IMPLEMENT `OracleInterface`

Category	Severity	Location	Status
Logical Issue	● Medium	contracts/oracles/BinanceOracle.sol (base): 11	● Resolved

Description

The contract `BinanceOracle` does not inherit the `OracleInterface` and does not properly implement it. This is because it takes an input of type `VBep20Interface` as opposed to the interface that takes an input `address` . The `ResilientOracle` interacts with the oracles via the `OracleInterface` , so that because the `BinanceOracle` does not properly implement it, it cannot be used in the `ResilientOracle` .

Recommendation

We recommend inheriting the interface to ensure that it properly implements it and so that it can be used in the `ResilientOracle` .

Alleviation

`[CertiK]` : The client made the recommended changes in commit: c4799d087ca2d1940854b7bee1c834fc2e59c2f9.

COV-01 | CHAINLINK CAN RETURN NEGATIVE PRICE THAT WILL NOT BE REVERTED

Category	Severity	Location	Status
Logical Issue	● Medium	contracts/oracles/ChainlinkOracle.sol (base): 206	● Resolved

Description

When calling `feed.latestRoundData()` , the return `answer` is a `int256` , allowing it to possibly be a negative number. Currently it is only checked if the `answer` is zero, allowing for negative prices to be cast as a `uint256` . We understand there is a low chance of a negative price being reported, which is why we give it a medium severity. However, if negative numbers are returned it can instead return the maximum price possible.

Recommendation

We recommend checking if the `answer` is less than or equal to zero and reverting if it is.

Alleviation

`[CertiK]` : The client made the recommended changes in commit: 78921ef20f98cc3acedd3d381ed6005b2fc2bd6e.

ROV-02 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Inconsistency	● Minor	contracts/ResilientOracle.sol (update1): 89	● Resolved

Description

In the `constructor()` of the contract `ResilientOracle`, the input `vaiAddress` is not checked to not be `address(0)`.

Recommendation

We recommend adding a check the passed-in address is not `address(0)` to prevent unexpected errors and remain consistent.

Alleviation

`[CertiK]` : The client made the recommended changes in commit: 8a1fe027d65bd298fdabfb139a253dc9b86d14fc.

COV-02 | INCORRECT EMIT EVENT

Category	Severity	Location	Status
Logical Issue	● Informational	contracts/oracles/ChainlinkOracle.sol (base): 92~93	● Resolved

Description

In the function `setDirectPrice` , the `prices[asset]` is set to be equal to `price` before the event is emitted. This then emits the event so that all three of the parameters `previousPriceMantissa` , `requestedPriceMantissa` , and `newPriceMantissa` will be the same.

Recommendation

We recommend storing the `previousPriceMantissa` in a temporary variable before assigning the input `price` and emitting this in the event.

Alleviation

`[CertiK]` : The client made the recommended changes in commit: d1bf4dc199ae87d6b5e4e1567ff41bdd11d0d6bb.

POV-02 | UNNECESSARY CASTING

Category	Severity	Location	Status
Inconsistency	● Informational	contracts/oracles/PythOracle.sol (update1): 146, 149	● Resolved

Description

In the function `getUnderlyingPrice()` , the input `vToken` is an `address` . Thus when comparing it to the addresses `vBnb` and `vai` it does not need to be cast as an `address` .

Recommendation

We recommend removing the unnecessary casting.

Alleviation

`[CertiK]` : The client made the recommended changes in commit: 0cc250e327adcfd0aa982d03c840266f0238f6e6.

ROV-01 | `fallbackPrice` IS TESTED AGAINST `mainPrice`

Category	Severity	Location	Status
Logical Issue	● Informational	contracts/ResilientOracle.sol (base): 195, 234	● Resolved

Description

In the function `getUnderlyingPrice()` , if the validation of the `mainPrice` vs. `pivotPrice` and `fallbackPrice` vs. `pivotPrice` fails. Then the `fallbackPrice` is tested against the `mainPrice` and if the validation passes, then the `mainPrice` is returned. However, if the `mainPrice` is being returned then it should be the price that is tested. Note that this goes against the comments at the beginning of the function that state the "fallback oracle against pivot oracle or main oracle".

Recommendation

We recommend determining if the `mainPrice` or `fallbackPrice` should be returned and have the call to `boundValidator.validatePriceWithAnchorPrice()` be consistent with this.

Alleviation

`[CertiK]` : The client changed the code so that it checks the `mainPrice` against the `fallbackPrice` in commit: 47ea8274691ca8f9a8fcd691dfe48e5269b4cfbc.

TOV-01 | MISSING CHECKS FOR FEEDS AND POOLS

Category	Severity	Location	Status
Logical Issue	● Informational	contracts/oracles/TwapOracle.sol (base): 153~165	● Resolved

Description

We understand that these are privileged functions and that these checks may be performed off chain, which is why we only mark this as informational.

The `TwapOracle` is only designed to handle pools with WBNB or BUSD and it should be checked that any `pancakePool` has one of these in its pair along with the asset, and that the bools `isBnbBased` and `isReversedPool` are set properly.

The `ChainlinkOracle` is designed to only use feeds for the assets it is assigned to along with USD. It should be checked that the feed assigned to an asset is for that asset and USD.

The `PythOracle` is designed to only use the `pythId` for the assets that it is assigned to along with USD. It should be checked that the feed assigned to an asset is for that asset and USD.

If the wrong values are used, then this can cause the wrong price to be returned by the oracle which may allow funds to be exploited from the protocol.

Recommendation

We recommend sharing the process of how new token configurations will be set to ensure that they will not be accidentally set with an incompatible value.

Alleviation

`[CertiK]` : The client stated the checks will be performed during the setup of a VIP. As this is out of scope of this audit, we will consider it as a black box and assume its functional correctness. We recommend carefully vetting the setup of each VIP to ensure these checks are made.

`[Venus]` : "We will delegate this check to the setup of the VIP. These configurations can only be done using the Governance process. Timelock contract will only have permission for these configurations."

VPB-01 | TYPOS AND INCONSISTENCIES

Category	Severity	Location	Status
Inconsistency	● Informational	contracts/ResilientOracle.sol (base): 115, 176; contracts/oracles/BoundValidator.sol (base): 145~148; contracts/oracles/ChainlinkOracle.sol (base): 183; contracts/oracles/PythOracle.sol (base): 130	● Resolved

Description

In the contract `ChainlinkOracle` :

- the comments above the function `_getChainlinkPrice()` uses "exit", when it should be "exist".

In the contract `ResilientOracle` :

- the comments above the function `setTokenConfigs()` misspells "length" as "lenght".
- the comment in the function `updatePrice()` , only mentions if the pivot oracle is the `PythOracle` the call will revert and need to be caught. However, if the pivot oracle is any oracle that is not a `TwapOracle` the call will revert and need to be caught.

In the contract `BoundValidator` :

- a storage gap is added and the comment above it mentions that this contract is designed to be inherited, however, this contract is not inherited. We recommend having this contract be inherited, removing the storage gap, or providing more information on how this contract is to be implemented in future iterations.

In the contract `PythOracle` :

- the comments above `getUnderlyingPrice()` state "@return price Underlying price with a precision of 10 decimals", however the precision is with 18 not 10 decimals.

Recommendation

We recommend fixing the typos and inconsistencies mentioned above.

Alleviation

`[CertiK]` : The client made all of the recommended changes in the following commits:

- d6747cca2e552afad4e8a7977beb7e516f71c93f;

- 9162a4900c21f83a743ae3ed2a2d4174e7e76ef0.

VPB-05 | ACCESS CONTROL CONVENTION

Category	Severity	Location	Status
Logical Issue	● Informational	contracts/ResilientOracle.sol (base): 143, 163; contracts/oracles/ChainlinkOracle.sol (base): 75; contracts/oracles/PythOracle.sol (base): 94	● Resolved

Description

In other audits you had clarified:

"The convention for function signatures in ACM checks is to use the signature equivalent to the one used in the selector computation (e.g. f(uint, VToken) becomes `f(uint256,address)`) to avoid ambiguities where multiple type names are possible. An exception to this convention is struct parameters; instead of listing the struct contents as a tuple, which is a canonical EVM type name for structs, we just use the struct name."

However the following do not seem to match this convention:

- the access check in the contract `ChainlinkOracle` for the function `setUnderlyingPrice()` uses `setUnderlyingPrice(VBep20Interface,uint256)` ;
- the access check in the contract `PythOracle` for the function `setUnderlyingPythOracle()` uses `setUnderlyingPythOracle(IPyth)` ;
- the access check in the contract `ResilientOracle` for the function `setOracle()` uses `setOracle(address,address,OracleRole)` . Where `OracleRole` is an `enum` and according to the solidity docs should be `uint8` ;
- the access check in the contract `ResilientOracle` for the function `enableOracle()` uses `enableOracle(address,OracleRole,bool)` . Where `OracleRole` is an `enum` and according to the solidity docs should be `uint8` ;

Recommendation

We recommend ensuring these match your conventions.

Alleviation

`[CertiK]` : The client changed the function signatures to match their convention in commit: cf682c76f381d9dbc07cb5f478e02c8602343875.

OPTIMIZATIONS | VENUS - ORACLE

ID	Title	Category	Severity	Status
BOV-02	Gas Optimizations When Comparing Strings	Gas Optimization	Optimization	● Resolved
COV-03	Can Use Custom Errors	Gas Optimization	Optimization	● Acknowledged
COV-04	Use Temporary Variable To Store `previousPriceMantissa`	Gas Optimization	Optimization	● Resolved
POV-01	Can Use Modifier To Zero Check Address	Gas Optimization	Optimization	● Resolved
VPB-02	Unnecessary Checks	Gas Optimization	Optimization	● Resolved
VPU-01	Unchecked Blocks Can Optimize Contract	Gas Optimization	Optimization	● Resolved

BOV-02 | GAS OPTIMIZATIONS WHEN COMPARING STRINGS

Category	Severity	Location	Status
Gas Optimization	● Optimization	contracts/oracles/BinanceOracle.sol (base): 47	● Resolved

Description

In the function `setMaxStalePeriod()`, instead of using `compare()` to determine if the input `symbol` is empty. The string can be converted to bytes and the length of the bytes checked. This will save around 809 gas on successful calls and 712 gas on calls that will revert due to this check.

In the function `compare()`, the string can be cast into `bytes` directly as opposed to using `abi.encodePacked`, which costs more gas. In addition, `abi.encodePacked` may be removed in future solidity versions (see this issue: https://github.com/ethereum/solidity/issues/11593).

Recommendation

We recommend implementing the gas optimizations above.

Alleviation

`[CertiK]`: The client made the recommended changes in commit: d7e15884f5f6a2def4875b775d203e0b90bbac05.

COV-03 | CAN USE CUSTOM ERRORS

Category	Severity	Location	Status
Gas Optimization	● Optimization	contracts/oracles/ChainlinkOracle.sol (base): 49, 104, 133, 206, 207, 210	● Acknowledged

Description

From Solidity `v0.8.4` , there are more gas-efficient ways to explain to users why an operation failed than through strings. Using custom errors can significantly reduce the size of the deployed bytecode and reduce the gas cost when calls revert.

We give the relevant lines in `ChainlinkOracle` , however, this applies to all files.

Recommendation

We recommend considering the use of custom errors to reduce gas costs. For more information see: https://blog.soliditylang.org/2021/04/21/custom-errors/.

Alleviation

`[CertiK]` : The client acknowledged the finding and stated they would implement custom errors in the future and will not be part of this audit engagement.

COV-04 | USE TEMPORARY VARIABLE TO STORE `previousPriceMantissa`

Category	Severity	Location	Status
Gas Optimization	● Optimization	contracts/oracles/ChainlinkOracle.sol (base): 78~79	● Resolved

Description

In the function `setUnderlyingPrice()` , the value of `prices[asset]` can be stored in a temporary variable before the input `underlyingPriceMantissa` and the temporary variable can be used as the `previousPriceMantissa` in the emitted event. This allows the event to be emitted after the assignment and saves a small amount of gas.

Recommendation

We recommend storing the `previousPriceMantissa` as a temporary variable and using the temporary variable in the emitted event to save gas and remain consistent.

Alleviation

`[CertiK]` : The client made the recommended changes in commit: 58b3b2ff26fdcdbdaf6522fb4ffe6ebe71331af0.

POV-01 | CAN USE MODIFIER TO ZERO CHECK ADDRESS

Category	Severity	Location	Status
Gas Optimization	● Optimization	contracts/oracles/PythOracle.sol (base): 107, 136	● Resolved

Description

In the contract `PythOracle` :

- the `initialize()` function checks that `underlyingPythOracle_` is not the zero address, when the `notNullAddress` modifier can be used to be more consistent and decrease the contracts deployment size.
- the function `getUnderlyingPrice` also checks that `underlyingPythOracle` is not the zero address, when the `notNullAddress` modifier can be used to be more consistent and decrease the contracts deployment size.

While this will increase the gas costs on function calls, we recommend remaining consistent throughout the codebase.

Recommendation

We recommend using the modifier to be consistent and decrease the deployment size of the contract.

Alleviation

`[CertiK]` : The client made the recommended changes in commit: 8439068499515f40cabc8ed5f1d6632dd621826f.

VPB-02 | UNNECESSARY CHECKS

Category	Severity	Location	Status
Gas Optimization	● Optimization	contracts/ResilientOracle.sol (base): 118, 122, 267; contracts/oracles/BoundValidator.sol (base): 59, 64, 86; contracts/oracles/ChainlinkOracle.sol (base): 103, 107, 131; contracts/oracles/PythOracle.sol (base): 76, 80, 120; contracts/oracles/TwapOracle.sol (base): 104, 108, 153, 156	● Resolved

Description

In the contracts `ChainlinkOracle` , `PythOracle` , `TwapOracle` , and `ResilientOracle` :

The function `setTokenConfigs()` checks that the `msg.sender` is allowed to call it. However, it then calls the function `setTokenConfig()` which will check that the `msg.sender` is allowed to call it. As permission was given to call `setTokenConfigs()` the permission does not need checked to call `setTokenConfig()` .

Similarly this occurs in the contract `BoundValidator` for the function `setValidateConfigs()` .

In the contract `TwapOracle` :

The function `setTokenConfig()` makes the following checks:

```
 if (config.baseUnit == 0) revert("base unit must be positive");
 if (config.baseUnit != 10 ** IERC20Metadata(config.asset).decimals()) revert("base
unit decimals must be same as asset decimals");
```

However `10 ** IERC20Metadata(config.asset).decimals()` cannot be 0, so that checking if `config.baseUnit` is zero is unnecessary.

Recommendation

We recommend refactoring the code so that it only checks that the `msg.sender` is allowed to call `setTokenConfigs()` to avoid unnecessary checks. In addition, we recommend removing the unnecessary check in `setTokenConfig()` .

Alleviation

`[CertiK]` : The client made the recommended changes in commit: 81dc384e076b3f42b70912d144b2b9d7d36d0d5b.

With this new implementation being given access to `setTokenConfig()` or `setValidateConfig()` also gives access to call the functions `setTokenConfigs()` or `setValidateConfigs()` respectively.

VPU-01 | UNCHECKED BLOCKS CAN OPTIMIZE CONTRACT

Category	Severity	Location	Status
Gas Optimization	● Optimization	contracts/oracles/BinanceOracle.sol (base): 109; contracts/oracles/BoundValidator.sol (base): 63~65; contracts/oracles/ChainlinkOracle.sol (base): 106~108, 209; contracts/oracles/PythOracle.sol (base): 79~81; contracts/oracles/TwapOracle.sol (base): 107~109, 214~217, 266~270	● Resolved

Description

The following operations can be placed in unchecked blocks as they cannot overflow or underflow.

In the contract `BinanceOracle` :

```
109  uint256 deltaTime = block.timestamp - updatedAt;
```

can be placed in an unchecked block as the following check is made beforehand.

```
107  if (block.timestamp < updatedAt) revert("updatedAt exceeds block time");
```

This ensures that `block.timestamp >= updatedAt` so that the subtraction cannot underflow.

In the contract `ChainlinkOracle` :

```
209  int256 deltaTime = block.timestamp - updatedAt;
```

can be placed in an unchecked block as the following check is made beforehand.

```
207  if (block.timestamp < updatedAt) revert("updatedAt exceeds block time");
```

This ensures that `block.timestamp >= updatedAt` so that the subtraction cannot underflow.

In the contract `TwapOracle` :

```
217  uint256 timeElapsed = block.timestamp - oldTimestamp;
```

can be placed in an unchecked block as the following check is made beforehand.

```
215  if (block.timestamp < oldTimestamp) revert("now must come after before");
```

This ensures that `block.timestamp >= oldTimestamp` so that the subtraction cannot underflow.

In general, `for` loops can have the increment placed in the body of the loop inside an unchecked block.

Recommendation

We recommend placing the mentioned code in unchecked blocks to save gas.

Alleviation

`[CertiK]` : The client made the recommended changes in commit: e3dd0e4a78390cc718abcaf977b8281cb2ba7a71.

APPENDIX | VENUS - ORACLE

Finding Categories

Categories	Description
Centralization / Privilege	Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.
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.
Logical Issue	Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.
Inconsistency	Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

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.