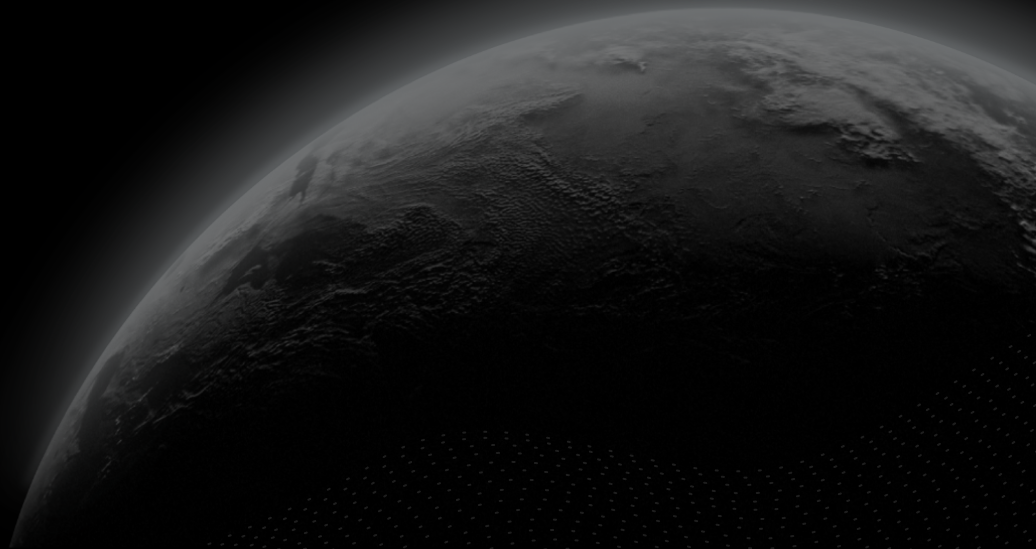




Security Assessment

Venus - Liquidator Contract

CertiK Assessed on Jul 4th, 2023





CertiK Assessed on Jul 4th, 2023

Venus - Liquidator Contract

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

Executive Summary

TYPES

DeFi

ECOSYSTEM

Binance Smart Chain
(BSC)

METHODS

Manual Review, Static Analysis

LANGUAGE

Solidity

TIMELINE

Delivered on 07/04/2023

KEY COMPONENTS

N/A

CODEBASE

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

View All in Codebase Page

COMMITTS

base: [9384b5256f48efcb5e85d085e930e51a371cfca3](#)update1: [a2e94c94e0aedef478a8bcee1efdecbb1958bbe20c](#)update2: [90bcafe9ce81135b62ae08d984ea4e00b22cc464](#)

View All in Codebase Page

Vulnerability Summary



9

Total Findings

7

Resolved

2

Mitigated

0

Partially Resolved

0

Acknowledged

0

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.

1 Medium

1 Resolved



Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.

2 Minor

2 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.

4 Informational

4 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 - LIQUIDATOR CONTRACT

I Summary

[Executive Summary](#)

[Vulnerability Summary](#)

[Codebase](#)

[Audit Scope](#)

[Approach & Methods](#)

I Summary

[Liquidator](#)

I Out Of Scope Dependencies

I Findings

[LLV-04 : Centralization Risks in Liquidator.sol](#)

[LLV-05 : Centralized Control of Contract Upgrade](#)

[LLV-03 : If `vToken` Is Added Multiple Times To `pendingRedeem` Array Multiple Copies Will Not Be Removed](#)

[LLV-08 : Possible Reentrancy](#)

[LLV-09 : Lack Of Input Validation](#)

[LLP-01 : `markets\(\)` Does Not Use All Return Values](#)

[LLV-06 : Unused Custom Error](#)

[LLV-07 : Typos](#)

[LLV-10 : Multiple Functions To Set Access Control Manager](#)

I Optimizations

[LLV-01 : Inefficient For Loop](#)

[VPB-01 : Unchecked Blocks Can Optimize Contract](#)

I Appendix

I Disclaimer

CODEBASE | VENUS - LIQUIDATOR CONTRACT

Repository

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

Commit

base: [9384b5256f48efcb5e85d085e930e51a371cfca3](#)

update1: [a2e94c94e0aedd478a8bcee1efdecb1958bbe20c](#)



update2: [90bcafe9ce81135b62ae08d984ea4e00b22cc464](#)

update3: [665119148c76291f8a72d778b2e4140c7c80456c](#)

AUDIT SCOPE | VENUS - LIQUIDATOR CONTRACT

2 files audited ● 1 file with Mitigated findings ● 1 file without findings



ID	Repo	Commit	File	SHA256 Checksum
● LLV	VenusProtocol/venus-protocol	9384b52	 Liquidator.sol	0cf24d136810e300f9baa1db862b16b9f5e48b4fc137ed84f7af4f3945078a86
● LSL	VenusProtocol/venus-protocol	9384b52	 LiquidatorStorage.sol	cd3f089b2f4cebad73746c8ae007bf17289bdf231fae74ee47378e2cb871d608

APPROACH & METHODS | VENUS - LIQUIDATOR CONTRACT

This report has been prepared for Venus to discover issues and vulnerabilities in the source code of the Venus - Liquidator Contract 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 - LIQUIDATOR CONTRACT

Liquidator

This contract is designed to add restrictions to liquidations on the Venus Protocol. It has the functionality to add an allowlist for any borrower, which only allows those on the allowlist to liquidate their borrows. In addition, it has logic to ensure that VAI borrows are liquidated before any others if they are above a minimum threshold. During the liquidation a portion of the seized `vToken` is sent to the liquidator and the remaining is attempted to be redeemed. If it is successfully redeemed, then the underlying that was redeemed for is transferred to the protocol share reserve. If the redemption fails, then it is added to a pending redeem array. Those `vTokens` held by the contract that is pending redemption and stored in the `pendingRedeem` array then have their redemption attempted pseudo-automatically when `liquidateBorrow()` is called or directly if `reduceReserves()` is called.

OUT OF SCOPE DEPENDENCIES

VENUS - LIQUIDATOR CONTRACT

The protocol is serving as the underlying entity to interact with out-of-scope dependencies. The out-of-scope dependencies that the contracts interact with are:

- `vTokens` including `vBnb` ;
- `comptroller` ;
- `vaiController` ;
- `wBNB` ;
- `protocolShareReserve` ;

The scope of the audit treats out-of-scope dependencies as black boxes and assumes their functional correctness.

In addition, it is assumed that the `protocolShareReserve` that the contracts interact with will have the functionality introduced in this PR: <https://github.com/VenusProtocol/protocol-reserve/pull/2>.

FINDINGS | VENUS - LIQUIDATOR CONTRACT



9
Total Findings

0
Critical

2
Major

1
Medium

2
Minor

4
Informational

This report has been prepared to discover issues and vulnerabilities for Venus - Liquidator Contract . Through this audit, we have uncovered 9 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:

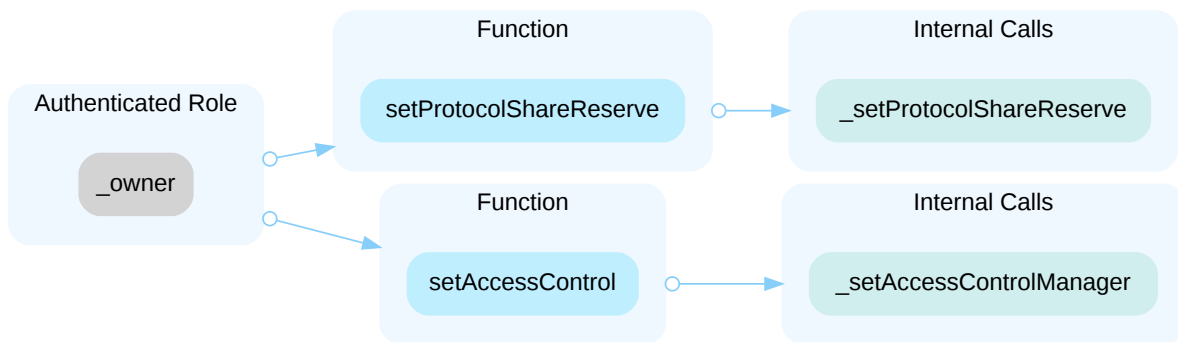
ID	Title	Category	Severity	Status
LLV-04	Centralization Risks In Liquidator.Sol	Centralization	Major	● Mitigated
LLV-05	Centralized Control Of Contract Upgrade	Centralization	Major	● Mitigated
LLV-03	If <code>vToken</code> Is Added Multiple Times To <code>pendingRedeem</code> Array Multiple Copies Will Not Be Removed	Logical Issue	Medium	● Resolved
LLV-08	Possible Reentrancy	Volatile Code	Minor	● Resolved
LLV-09	Lack Of Input Validation	Volatile Code	Minor	● Resolved
LLP-01	<code>markets()</code> Does Not Use All Return Values	Coding Style	Informational	● Resolved
LLV-06	Unused Custom Error	Coding Issue	Informational	● Resolved
LLV-07	Typos	Coding Style	Informational	● Resolved
LLV-10	Multiple Functions To Set Access Control Manager	Coding Issue	Informational	● Resolved

LLV-04 | CENTRALIZATION RISKS IN LIQUIDATOR.SOL

Category	Severity	Location	Status
Centralization	● Major	Liquidator.sol (base): 211 , 225 , 350 , 534	● Mitigated

Description

In the contract `Liquidator` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and change the `_accessControlManager` which can allow them to give any of the privileges outline below or the `protocolShareReserve` to collect the protocols share for themselves.



In the contract `Liquidator`, the `_accessControlManager` contract can give privilege to the following functions:

- `restrictLiquidation()`, which can be used to prevent a borrower from being liquidated by having no liquidators on their allowlist.
- `unrestrictLiquidation()`, which can be used to allow a non-allowlisted user to liquidate an account.
- `addToAllowlist()`, which can allow any address to liquidate an account.
- `removeFromAllowlist()`, which can remove an liquidators ability to liquidate an account. If all liquidators are removed from a borrowers allowlist this will make it so they cannot be liquidated.
- `setTreasuryPercent()`, which can be set so that the protocol share reserve receives none of the liquidation incentive. Or it can be set so that it receives all of the liquidation incentive.
- `setMinLiquidatableVAI()`, which can be set to a small value so that a liquidator would always need to liquidate VAI before they could liquidate any other borrows.
- `setPendingRedeemChunkLength()`, which can be set to a large value and cause a denial of service if the `pendingRedeem` array is too large. Or set to zero to prevent any tokens pending redemption from being redeemed.
- `pauseForceVAILiquidate()`, which can be used to not have to liquidate VAI borrows before others.
- `resumeForceVAILiquidate()`, which can be used to ensure that VAI borrows are liquidated before any others.

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 (2/3, 3/5) 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, 06/27/3023] : The owner of the proxy Liquidator contract (0x0870793286aada55d39ce7f82fb2766e8004cf43) 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.

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 every mentioned function. Moreover, we'll allow [a] (Fast-track) and [b] (Critical) also to execute the following functions:

- restrictLiquidation(address)
- addToAllowlist(address,address)
- pauseForceVAILiquidate()
- resumeForceVAILiquidate()

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

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

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

LLV-05 | CENTRALIZED CONTROL OF CONTRACT UPGRADE

Category	Severity	Location	Status
Centralization	● Major	Liquidator.sol (base): <u>73</u>	● Mitigated

Description

`Liquidator.sol` is an upgradeable contract. 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.

Recommendation

We recommend that the team make efforts to restrict access to the admin of the proxy contract. A strategy of combining a time-lock and a multi-signature (2/3, 3/5) wallet can be used to prevent a single point of failure due to a private key compromise. In addition, the team should be transparent and notify the community in advance whenever they plan to migrate to a new implementation contract.

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

Short Term:

A combination of a time-lock and a multi signature (2/3, 3/5) wallet mitigate the risk by delaying the sensitive operation and avoiding a single point of key management failure.

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

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

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

Long Term:

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

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

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

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

Permanent:

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

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

Note: we recommend the project team consider the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

Alleviation

[Venus, 06/27/3023] : The current admin of the Liquidator proxy contract [1] is the ProxyAdmin contract deployed at 0x2b40B43AC5F7949905b0d2Ed9D6154a8ce06084a.

The owner of this ProxyAdmin contract is 0x939bd8d64c0a9583a7dcea9933f7b21697ab6396, the Normal Timelock 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, this contract will be upgraded only via a Normal VIP, involving the community in the process.

[1] <https://bscscan.com/address/0x0870793286aada55d39ce7f82fb2766e8004cf43>

LLV-03 | IF `vToken` IS ADDED MULTIPLE TIMES TO `pendingRedeem` ARRAY MULTIPLE COPIES WILL NOT BE REMOVED

Category	Severity	Location	Status
Logical Issue	● Medium	Liquidator.sol (base): 368~369 , 412~414	● Resolved

Description

If the redemption of a `vToken` fails when liquidating, then that `vToken` is added to the `pendingRedeem` array. Thus if multiple liquidations happen with the same `vTokenCollateral` and all redemptions fail, the same `vToken` will be added to the `pendingRedeem` array multiple times. Thus when `_reduceReservesInternal()` is called it will reference the same `vToken` at different indexes. However, this function uses the contracts balance of `vToken`, so that if it redeems for one of the indexes the `vTokenBalance` of the contract will become zero. Then when any of the other indexes are attempted, the redemption will revert as it will have a zero amount. In this way the `pendingRedeem` array will grow with no ability to remove some of its elements, which can be used to prevent the contract from being able to redeem `vToken` and transfer the underlying asset to the protocol share reserve.

Scenario

Assume that any call to `redeem()` on `vExample` will revert, for example if redemptions are paused for that market. Now assume that there are two borrows eligible for liquidation that have `vExample` as collateral and that a user then calls `liquidateBorrow()` to liquidate both of these borrows seizing `vExample`. When `_distributeLiquidationIncentive()` is called the liquidators `vExample` will be transferred to them, but in both cases the attempt to redeem the underlying will revert, causing `vExample` to be pushed to the `pendingRedeem` array. For simplicity assume that `pendingRedeem[0]` and `pendingRedeem[1]` are `vExample`.

Now assume that a period of time goes by and that redemptions for `vExample` will succeed, for example the redemptions are unpaused, and that no more elements have been added to the `pendingRedeem` array. Then assume either `liquidateBorrow()` or `reduceReserves()` is called so that `_reduceReservesInternal()` is called.

It will start with `pendingRedeem[1]` and redeem the entire contracts balance of `vExample`. As it successfully redeemed it will remove `pendingRedeem[1]` from the array and the contracts balance of `vExample` will be 0. Next it will reach `pendingRedeem[0]` and attempt to redeem the entire contracts balance of `vExample`. However, the balance will be 0 causing this to revert due to the following check in `vToken`:

```
require(redeemTokensIn == 0 || redeemAmountIn == 0, "one of redeemTokensIn or redeemAmountIn must be zero");
```

`_redeemUnderlying()` will catch this revert and return false so that it is **not** removed from the array. Thus this element will remain in the array indefinitely unless `vExample` tokens are sent directly to the contract.

Recommendation

We recommend either checking if the `vTokenBalance` is zero and removing it from the pending redeem in this case, or only adding a `vToken` to the `pendingRedeem` array when the received amount of `vToken` is nonzero and is not already included in the pending redeem array.

Alleviation

[Certik, 06/27/3023]: The client made the recommended changes in commits:

- [c831054531b10ca1c7cd54f2f4549e9d4c84264f](#);
- [a2e94c94e0aedef478a8bcee1efdecb1958bbe20c](#);

In addition, the check that the `vTokenCollateral` is a listed market ensures that the maximum size of the `pendingRedeem` array is at most the total number of listed markets. Thus it can be ensured that transactions do not run out of gas by keeping the total number of listed markets sufficiently small. Considering this depends on future actions, we consider this outside the scope of this audit and mark this finding as *resolved*. However, we recommend when listing new markets to ensure that it will not cause `liquidateBorrow()` to become close to the gas limit.

LLV-08 | POSSIBLE REENTRANCY

Category	Severity	Location	Status
Volatile Code	● Minor	Liquidator.sol (base): <u>354~359</u>	● Resolved

Description

The functions `liquidateBorrow()` and `reduceReserves()` are external and can be called by anyone. While `liquidateBorrow()` has the `nonReentrant` modifier, `reduceReserves()` does not and may allow an attacker to re-enter the contract. In particular, the external calls made to the inputs `vToken` and `vTokenCollateral` of `liquidateBorrow()` may be used to call `reduceReserves()` and re-enter the contract.

This finding is considered minor because the reentrancy only causes out-of-order events.

Recommendation

We recommend adding the `nonReentrant` modifier to all user facing functions.

Alleviation

[Certik, 06/27/3023]: The client made the recommended changes in commit: 49f76b28f618f3d110708981b63595843171c87d.

LLV-09 | LACK OF INPUT VALIDATION

Category	Severity	Location	Status
Volatile Code	Minor	Liquidator.sol (base): 297 , 300	Resolved

Description

The function `liquidateBorrow()` takes an input `vToken` and `vTokenCollateral` which are never checked to be supported `vToken`. Thus a malicious user can input a malicious contracts address for them so that it will fail on redemption and `vTokenCollateral` will be added to the `pendingRedeem` array. This leads to a potential denial of service as enough `vToken` can be added to the array so that other pending redemptions will not be executed. In addition, this allows an entry point for a hacker as whenever `liquidateBorrow()` is called, it will call `_reduceReservesInternal()` which will make an external call to a potentially malicious address.

Recommendation

We recommend verifying that the input `vToken` and `vTokenCollateral` are supported.

Alleviation

[Certik, 06/28/2023]: The client made the recommended changes in commit: [90bcafe9ce81135b62ae08d984ea4e00b22cc464](#).

LLP-01 | `markets()` DOES NOT USE ALL RETURN VALUES

Category	Severity	Location	Status
Coding Style	● Informational	Liquidator.sol (update2): 29	● Resolved

Description

The function `markets()` in the comptroller returns a triple `(bool, uint256, bool)`, however the interface only has a `bool` return value. Currently this will function as intended as it will reference the correct memory to return the first `bool` in the triple which corresponds to if the market is listed. However, it is possible that in future iterations of the virtual machine that this behavior will not remain the same.

Recommendation

We recommend including all the return values of the function in the interface.

Alleviation

[Certik, 07/04/2023]: The client made the recommended changes in commit: [665119148c76291f8a72d778b2e4140c7c80456c](#).

LLV-06 | UNUSED CUSTOM ERROR

Category	Severity	Location	Status
Coding Issue	● Informational	Liquidator.sol (base): <u>172~173</u>	● Resolved

Description

The custom error `UnderlyingTransferFailed` is declared but never used.

Recommendation

We recommend either implementing or removing the unused custom error.

Alleviation

[Certik, 06/27/3023]: The client made the recommended changes in commit: 925c44d6840d68e2388483e1e3b3eeb57baf37bd.

LLV-07 | TYPOS

Category	Severity	Location	Status
Coding Style	● Informational	Liquidator.sol (base): <u>405</u> , <u>407</u>	● Resolved

Description

In the function `_distributeLiquidationIncentive()`, `siezedAmount` should be spelled as `seizedAmount`.

Recommendation

We recommend fixing the typos mentioned above.

Alleviation

[Certik, 06/27/3023]: The client made the recommended changes in commit: [9555b7ef5937432c7893e93753ee70d693bf1d75](https://github.com/certiklabs/venus-liquidator/commit/9555b7ef5937432c7893e93753ee70d693bf1d75).

LLV-10 | MULTIPLE FUNCTIONS TO SET ACCESS CONTROL MANAGER

Category	Severity	Location	Status
Coding Issue	● Informational	Liquidator.sol (base): 529~537	● Resolved

Description

The function `setAccessControl()` and the inherited function `setAccessControlManager()` for `AccessControlledV8` both call the inherited internal function `_setAccessControlManager` to set the `_accessControlManager`.

Recommendation

We recommend removing `setAccessControl()` as the inherited `setAccessControlManager()` has the same functionality.

Alleviation

[Certik, 06/27/3023]: The client made the recommended changes in commit: [f79a308b2c9ba7d0b684bbf8f90748e665b6f4f0](#).

OPTIMIZATIONS | VENUS - LIQUIDATOR CONTRACT

ID	Title	Category	Severity	Status
<u>LLV-01</u>	Inefficient For Loop	Gas Optimization	Optimization	● Resolved
<u>VPB-01</u>	Unchecked Blocks Can Optimize Contract	Gas Optimization	Optimization	● Resolved

LLV-01 | INEFFICIENT FOR LOOP

Category	Severity	Location	Status
Gas Optimization	● Optimization	Liquidator.sol (base): <u>367~379</u>	● Resolved

Description

The function `_reduceReservesInternal()` has a for loop that starts with the `index` at `range - 1` and decreases by 1 as long as the `index` is greater than or equal to 0. Thus at the final step the `index = 0` is decremented so that it must be a signed integer. However, in the logic of the for loop, the index is only used when it is an unsigned integer and the loop can be refactored to only use unsigned integers.

Recommendation

We recommend refactoring to avoid unnecessary casting between unsigned and signed integers. For example, the index can instead start from `range` and decrease by 1 as long as `index > 0`. Then `index - 1` can be used inside the logic as opposed to `index`.

Alleviation

[Certik, 06/27/3023]: The client made the recommended changes in commits:

- [6d33c87e5cdd4ee2b97b2fd4f913af731650cf63](#);
- [a2e94c94e0aedd478a8bcee1efdec1958bbe20c](#);

VPB-01 | UNCHECKED BLOCKS CAN OPTIMIZE CONTRACT

Category	Severity	Location	Status
Gas Optimization	● Optimization	Liquidator.sol (base): 367 ; Liquidator.sol (update1): 414	● Resolved

Description

In the function `reduceReservesInternal()`, the for loop `index` can be decremented inside an unchecked block to save gas as `index >= 0` ensures it will not underflow.

Recommendation

We recommend adding these unchecked blocks to save gas.

Alleviation

[Certik, 06/28/3023]: The client made the recommended changes in commits:

- [ead535434acc71f932158bce3c4408f89539389f](#);
- [aee7e2b2a7f6f676086663a0e603e68b35abea1a](#).

APPENDIX | VENUS - LIQUIDATOR CONTRACT

Finding Categories

Categories	Description
Gas Optimization	Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.
Coding Style	Coding Style findings may not affect code behavior, but indicate areas where coding practices can be improved to make the code more understandable and maintainable.
Coding Issue	Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.

Checksum Calculation Method

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

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

DISCLAIMER | CERTIK

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

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

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

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

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

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

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

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

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

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

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

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

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

CertiK | Securing the Web3 World

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

