Venus Token Converter Audit

OPENZEPPELIN SECURITY | OCTOBER 27, 2023

Security Audits

Table of Contents

Summary

Type
: DeFi

Timeline
: From 2023-09-11
: To 2023-09-20

Languages
: Solidity

Total Issues
: 20 (19 resolved)

Critical Severity Issues
: 0 (0 resolved)

High Severity Issues
: 0 (0 resolved)

Medium Severity Issues
: 2 (2 resolved)

Low Severity Issues
: 8 (7 resolved)

Notes & Additional Information
: 10 (10 resolved)

Scope

We audited the VenusProtocol/protocol-reserve repository at commit ca8f8ba.

In scope were the following contracts:

```
|   ├── RiskFundStorage.sol
|   ├── RiskFundV2.sol
|   └── XVSVaultTreasury.sol
├── TokenConverter
|   ├── AbstractTokenConverter.sol
|   ├── IAbstractTokenConverter.sol
|   ├── RiskFundConverter.sol
|   └── XVSVaultConverter.sol
└── Utils
    ├── Constants.sol
    └── Validators.sol
```

System Overview

The Venus protocol collects revenues from reserve interest and liquidations. The revenues are generated from the core lending pool as well as various isolated pools, and initially sent to the protocol share reserve contract. The function of the protocol share reserve is then to distribute these revenues to different targets, keeping track of which pools they were collected from.

Within the scope of this audit, these targets will include the `RiskFundConverter` and `XVSVaultConverter` contracts. These contracts are responsible for converting the received tokens into a specific single token type before sending them to different locations. The `RiskFundConverter` converts tokens to USDT before sending to the `RiskFundV2` contract and the `XVSVaultConverter` converts tokens to XVS before sending to the `XVSVaultTreasury`.

These conversions are meant to be performed in a distributed and efficient manner, and thus Venus will incentivize these conversions to be performed by offering discounts. By allowing external agents to provide USDT and XVS to the converter contracts to be swapped at a rate provided by the Venus `ResilientOracle` with a discount, users can benefit from the favorable swap rate which provides an arbitrage opportunity. Venus benefits by not being exposed to slippage and sandwich attacks if the conversions were to happen on an external AMM due to potentially large trade sizes.

`RiskFundConverter` and `XVSVaultConverter`. Extending Venus' `AccessControlV8` contract, it integrates with the `AccessControlManager` (ACM, out of scope) with permissioned functions handled by a governance timelock and it includes an owner for more critical operations.

Authorized users can set `convertConfigurations` which are unique for each `tokenAddressIn`, `tokenAddressOut` pair, setting an incentive for conversions of the pair. The incentive can be at most 50% as enforced by the `MAX_INCENTIVE` constant value. Conversion configurations are supported for any tokens, including fee-on-transfer types, for which the Venus `ResilientOracle` has a price. This provides flexibility, but does not allow the `XVSTokenConverter` and `RiskFundConverter` to strictly enforce a single `tokenAddressIn` as XVS or USDT, respectively.

The oracle address is set during initialization and can be changed by the owner thereafter. During conversions, no price validation is performed on the retrieved prices, and it is assumed that the oracle is not corrupted or malicious. The current implementation of the `ResilientOracle` collects prices from various sources to be robust and will cause a revert if any returned price is 0.

Security Model and Trust Assumptions

We trust that the Protocol Share Reserve handles all accounting and token distribution functions correctly. We trust the Resilient Oracle to provide reliable prices.

Privileged Roles

Those authorized in the ACM can perform the following privileged actions:

- Pause and resume conversions
- Set conversion configurations for any token pair
- Send funds from the `XVSVaultTreasury` to the `XVSVault`
- Configure direct transfers for pool/asset types in the `RiskFundConverter` (tokens get sent directly to risk fund with no conversion)

The owner can perform the following privileged actions:

- Sweep (transfer out) any amount of arbitrary tokens from the converter contracts
- Set the `convertibleBaseAsset`, `riskFundConverter`, and `shortfall` addresses in `RiskFundV2`
- Sweep any of the tracked assets from `RiskFundV2`
- Set the `XVSVault` address in `XVSVaultTreasury`

Fee-On-Transfer Tokens Lead to Improper Tracking

Untracked assets residing in `RiskFundConverter` can be tracked using the `updateAssetsState` function. If the asset is configured to be transferred directly to the destination (i.e., the `RiskFundV2` vault), the `poolAssetsFunds` in `RiskFundV2` will be updated using the `updatePoolState` call. However, if this asset is a fee-on-transfer token, the balance will not be properly tracked as the amount of funds sent from the `RiskFundConverter` will not be the same as the amount of funds received by the `RiskFundV2`. This could lead off-chain scripts designed to transfer funds from `RiskFundV2` to fail due to insufficient balances if they are reading from `poolAssetsFunds`. Consider tracking the actual amount of funds received by `RiskFundV2` instead of the amount of funds sent from `RiskFundConverter` to support fee-on-transfer tokens.

Update: *Resolved at commit 4025db0.*

ERC-777 Tokens Lead to Improper Tracking

`RiskFundConverter` and `XVSVaultConverter` are offering the income generated by the Venus protocol to external agents for either `USDT` or `XVS`. Upon swapping the assets, the agent can call, for example, the `convertExactTokens` function which transfers `USDT` or `XVS` from the agent to the converter contract and sends the tokens to the agent. By subtracting the token balance before the transfer by the token balance after the transfer, the converter contract knows how many tokens were sent out in order to update the internal state in the `postConversionHook` function.

If the token is an ERC-777 and the external agent has registered a `receive` hook, it will be able to execute arbitrary code during the transfer. During this arbitrary code execution, the agent is able to influence the token balance by either transferring out tokens or acquiring additional tokens (e.g., by interacting with a DEX or lending protocol). By performing this action, the agent is able to inflate or deflate the calculation of the amount of funds which were sent out of the converter contract.

As this value is used to keep track of the number of assets held by the converter, the agent is able to create a discrepancy between the amount of tokens that the converter is tracking and the one it is actually holding. In case the converter is tracking fewer tokens than it actually holds, the

~~reverts~~ as the converter does not hold enough tokens to perform a swap.

Consider tracking the amount of tokens which were sent out of the converter by subtracting the token balance of the converter instead of the token balance of the external agent.

Update: *Resolved at commit 939da0a.*

Low Severity

Inconsistent Convention for Checking Access Allowance

`_checkAccessAllowed` is called using the function signature (e.g., here and here), but it is called using a different convention in `XVSVaultTreasury` (variable name replaces type). Consider sticking to a consistent convention for `_checkAccessAllowed` to avoid the likelihood of checking for incorrect strings and causing functions to revert.

Update: *Resolved at commit b467b46.*

Inconsistent Zero-Address Checks

In the `RiskFundConverter` contract, there are some inconsistencies in how zero addresses are checked. For example:

- Three addresses are passed to the constructor but only the first is checked with `ensureNonzeroAddress`. `vBNB` and `NATIVE_WRAPPED` are never checked throughout the contract. Consider checking that all arguments are non-zero.
- Throughout the contract, zero addresses are checked using a mix of `ensureNonzeroAddress` and `!= address(0)`. Consider adopting a consistent convention.
- In `updateAssetsState`, `poolRegistry` is checked to be non-zero, but this seems unnecessary. It is not an argument to the function and is always checked when set. Consider removing the check in `updateAssetsState`, but adding it in the initializer.

Update: *Resolved at commit 0f58716.*

Owner Can Only Sweep Tracked Assets From Risk Fund

Consider adding arbitrary token sweeps to the risk fund to prevent donated tokens from becoming irrecoverable.

Update: *Resolved at commit a842667.*

Missing Docstrings

Throughout the codebase, there are several parts that do not have docstrings. For instance:

- Line 57 in `IAbstractTokenConverter.sol`
- Line 23 in `RiskFundStorage.sol`
- Line 274 in `RiskFundConverter.sol`
- Line 34 in `RiskFundStorage.sol`
- Line 36 in `XVSVaultConverter.sol`
- Line 84 in `XVSVaultTreasury.sol`
- Line 24 in `IAbstractTokenConverter.sol`
- Line 72 in `XVSVaultTreasury.sol`
- Line 18 in `IAbstractTokenConverter.sol`
- Line 22 in `IAbstractTokenConverter.sol`
- Line 48 in `IAbstractTokenConverter.sol`
- Line 32 in `IAbstractTokenConverter.sol`
- Line 56 in `IAbstractTokenConverter.sol`
- Line 40 in `IAbstractTokenConverter.sol`
- Line 20 in `IAbstractTokenConverter.sol`
- Line 55 in `RiskFundConverter.sol`
- Line 52 in `RiskFundConverter.sol`
- Line 58 in `RiskFundConverter.sol`

Consider thoroughly documenting all functions (and their parameters) that are part of any contract's public API. Functions implementing sensitive functionality, even if not public, should be clearly documented as well. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

- `RiskFundConverter` emits the `AssetsReservesUpdated` event upon updating the `poolsAssetsReserves` mapping in `updateAssetsState`. `poolsAssetsReserves` is updated in `updatePoolAssetsReserve` as well, but no event is emitted here.

- When tokens are sent from the `RiskFundConverter` to the `RiskFundV2` in `updateAssetsState`, the `AssetTransferredToDestination` event is emitted. Similarly, `_actualAmounts` in `AbstractTokenConverter` sends funds to `RiskFundV2`, but no event is emitted here.

Consider adding event emissions in the previously described cases to ensure that off-chain monitoring can properly track these occurrences.

Update: *Resolved at commit 35dc0a1.*

Missing Check of XVS Store Address

In the `XVSVaultTreasury` contract, the `xvsStore` address is obtained from the `xvsVault` before tokens are transferred to the `xvsStore`. However, there is no check that the address is non-zero.

Consider verifying the `xvsStore` address before sending tokens to it.

Update: *Resolved at commit e558b15.*

Sweeping Tokens in Risk Fund Should Be Protected by Access Control Manager

According to the `onlyOwner` vs. AccessControlManager (ACM) rule, more critical changes (i.e., relationships between contracts) should be protected with `onlyOwner` as this is protected by the normal timelock (24 hours of voting + 48 hours of delay). The ACM should be used for actions which should potentially bypass voting, enabling them to take the fast-track or critical route, or even to be executed directly through a multisig by guardians.

to be covered quickly in case of heavy market fluctuations.

Consider restricting access to the ACM instead of the `onlyOwner` modifier in this case.

Update: *Acknowledged, not resolved. The Venus team stated:*

> *After a discussion with the team, we came to the conclusion of using OnlyOwner for the sweepToken.*

Lack of Storage Gap

The upgradeable `XVSVaultConverter` contract does not have a storage gap configured. Storage gaps are a convention for reserving storage slots in a base contract, allowing future versions of that contract to use up those slots without affecting the storage layout of the child contract.

Consider adding a storage gap variable to avoid future storage clashes in upgradeable contracts.

Update: *Resolved at commit 2be108d.*

Notes & Additional Information

Incorrect Comments

Throughout the codebase, there are instances of misleading comments or comments with typographical errors. For example:

In `AbstractTokenConverter.sol`:

- line 240: "fater" should be "after"
- line 344: says `public` function, but has the `onlyOwner` modifier
- line 474: "liquity" should be "liquidity"
- line 143 and line 624: these functions set the destination address, not the oracle

In `RiskFundConverter.sol`:

Consider correcting these comments to improve the overall clarity of the codebase.

Update: *Resolved at commit 5d0f03b.*

Unnecessary Storage Usage in Conversion Configuration

The `convertConfigurations` mapping stores `ConversionConfig` structs for each `tokenAddressIn`, `tokenAddressOut` pair to keep track of incentives and check if those pairs are enabled. Since `tokenAddressIn`, `tokenAddressOut` are already used as keys in the mapping, and since `ConversionConfig` is not used anywhere else in the codebase, it is unnecessary to store the token addresses in the struct as well. Consider removing the token address fields from the `ConversionConfig` struct to avoid unnecessary storage usage and gas costs. Since the token addresses are only read from a struct in `setConversionConfig`, this would be the only function that would require refactoring after making this change.

Update: *Resolved at commit 91eb7a6.*

`postSweepToken` Should Revert Early on Insufficient Balance

`postSweepToken` in the `RiskFundConverter` contract is executed before residual tokens are transferred out. If the `amount` to sweep is greater than the balance in the contract, the function will thus revert after `postSweepToken`, which loops through the internal reserve accounting logic. A revert with division by zero can also occur in this case if attempting to sweep a token with 0 `assetReserves`. It is possible to catch both these cases early in `postSweepToken` without executing unnecessary logic.

Consider reverting early in `postSweepToken` to handle these cases and ensure more clearly defined error handling and gas efficiency.

Update: *Resolved at commit 1e367e1.*

Incorrect Error in `getAmountIn`

To get the amount of `tokenAddressIn` tokens a user should send to receive `amountOutMantissa` tokens of `tokenAddressOut`, users can call the `getAmountIn` function of `AbstractTokenConverter`. If the supplied `amountOutMantissa` is `0`, the

Consider correcting the error to accurately describe the situation.

Update: *Resolved at commit 8d6389c.*

Unused State Variables

Throughout the codebase, there are multiple unused state variables:

- The `maxLoopsLimit` state variable in the `MaxLoopsLimitHelpersStorage` contract.
- The `pancakeSwapRouter` state variable in the `RiskFundV1Storage` contract.
- The `minAmountToConvert` state variable in the `RiskFundV1Storage` contract.
- The `BLOCKS_PER_YEAR` state variable.

To improve the overall clarity, intentionality, and readability of the codebase, consider removing any unused state variables. In case the unused state variable is necessary to prevent storage collisions between different versions of a contract (e.g., `maxLoopsLimit`, `pancakeSwapRouter` and `minAmountToConvert`), consider changing the name of the variable to clarify that the variable is deprecated.

Update: *Resolved. The Venus team stated:*

> *The state variables `maxLoopsLimit`, `pancakeSwapRouter`, and `minAmountToConvert` already have explanations in their Natspec that they are deprecated. Regarding the `BLOCKS_PER_YEAR` state variable, it might have utility in the future. Importantly, it hasn't been imported in any file, ensuring that it does not impact the bytecode as of now.*

Unused Named Return Variables

Named return variables are a way to declare variables that are meant to be used within a function's body for the purpose of being returned as the function's output. They are an alternative to explicit in-line `return` statements.

Consider either using or removing any unused named return variables.

Update: *Resolved at commit 8227f4b.*

Lack of Security Contact

Providing a specific security contact (such as an email or ENS name) within a smart contract significantly simplifies the process for individuals to communicate if they identify a vulnerability in the code. This practice proves beneficial as it permits the code owners to dictate the communication channel for vulnerability disclosures, eliminating the risk of miscommunication or failure to report due to a lack of knowledge on how to do so. Additionally, if the contract incorporates third-party libraries and a bug surfaces in these, it becomes easier for the maintainers of those libraries to make contact with the appropriate person about the problem and provide mitigation instructions.

Throughout the codebase, there are contracts that do not have a security contact:

- The `IAbstractTokenConverter` contract
- The `ReserveHelpersStorage` contract
- The `RiskFundConverter` contract
- The `RiskFundV1Storage` contract
- The `AbstractTokenConverter` contract
- The `MaxLoopsLimitHelpersStorage` contract
- The `XVSVaultConverter` contract
- The `XVSVaultTreasury` contract
- The `RiskFundV2Storage` contract
- The `RiskFundV2` contract

Consider adding a NatSpec comment containing a security contact on top of the contracts' definition. Using the `@custom:security-contact` convention is recommended as it has been adopted by the OpenZeppelin Wizard and the ethereum-lists.

Update: *Resolved at commit 08bacb3.*

Constants Not Using `UPPER_CASE` Format

- The `corePoolComptroller` constant declared on line 20 in `RiskFundConverter.sol`
- The `vBNB` constant declared on line 24 in `RiskFundConverter.sol`

According to the Solidity Style Guide, constants should be named with all capital letters with underscores separating words. For better readability, consider following this convention.

Update: *Resolved at commit bda9bea.*

The Function _disableinitializers() Is Not Being Called in the Constructors of Multiple Initializable Contracts

An implementation contract in a proxy pattern allows anyone to call its `initialize` function. While not a direct security concern, preventing the implementation contract from being initialized is important, as this could allow an attacker to take over the contract. This would not affect the proxy contract's functionality, as only the implementation contract's storage would be affected.

Throughout the codebase, there are multiple initializable contracts where `_disableInitializers()` is not called in the constructor. For instance:

- The initializable contract `RiskFundConverter` within the `RiskFundConverter.sol` file.
- The initializable contract `XVSVaultConverter` within the `XVSVaultConverter.sol` file.

Consider calling `_disableInitializers()` in initializable contract constructors to prevent malicious users from tampering with implementation storage.

Update: *Resolved at commit 78150be.*

Lack of Indexed Event Parameters

Within `AbstractTokenConverter.sol`, several events do not have their parameters indexed. For instance:

- line 51

Consider indexing event parameters to improve the ability of off-chain services to search for and filter for specific events.

Update: *Resolved at commit 3876d3b.*

Conclusions

The Token Converter scope is well-implemented, with only 2 issues of medium severity and several issues of low or note severity. All in all, the codebase is in a good state. However, we present several recommendations on how to improve the maturity of the codebase.

Related Posts

Beefy Zap Audit

BeefyZapRouter serves as a versatile intermediary designed to execute users' orders through routes...

Security Audits

OpenBrush Contracts Library Security Review

OpenBrush is an open-source smart contract library written in the Rust programming language and the...

Security Audits

Linea Bridge Audit

Linea is a ZK-rollup deployed on top of Ethereum. It is designed to be EVM-compatible and aims to...

Security Audits