OpenZeppelin | security

Venus Private Conversions Audit

February 20, 2024

Table of Contents

Summary

Type	DeFi
Timeline	From 2023-11-27 To 2023-12-04
Languages	Solidity

Total Issues	13 (12 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	1 (1 resolved)
Low Severity Issues	4 (4 resolved)
Notes & Additional Information	7 (7 resolved)

Scope

We audited the VenusProtocol/protocol-reserve repository at commit dd07dae.

In scope were the following contracts:

```
contracts
├── Interfaces
│   └── IConverterNetwork.sol
├── TokenConverter
│   ├── AbstractTokenConverter.sol
│   ├── ConverterNetwork.sol
│   ├── IAbstractTokenConverter.sol
│   ├── RiskFundConverter.sol
│   └── SingleTokenConverter.sol
└── Utils
    └── ArrayHelpers.sol
```

System Overview

Revenues generated from the Venus protocol are initially sent to a protocol share reserve contract, which is then responsible for distributing the various tokens to different converter contracts. Each converter can perform a set amount of token conversions, accepting a particular base asset in exchange for the tokens that are distributed to it. As an incentive for users to perform these conversions, tokens can be bought with the designated base asset at a discounted price. This allows conversions to be performed in a decentralized manner while protecting Venus from having to convert tokens on the open market, which may encounter the risk of manipulation with potentially large trade sizes.

Once conversions take place, converter contracts will transfer the received base assets to their corresponding destination address. When several converter contracts offer complementary conversions, the process can be optimized. For example, the `RiskFundConverter` contract may offer XVS to USDT conversions, while the `SingleTokenConverter` contract may offer USDT to XVS conversions. The goal is to match these conversions, removing the incentives for external users. This process is called private conversion in the current system and is initiated by the protocol share reserve contract which calls `updateAssetsState` after releasing funds to any converter contract. Afterwards, the converter contract will query its `ConverterNetwork` contract to retrieve other converters on the network which may be able to perform the desired conversion immediately. The converter will then try to match and convert the maximum possible amount of tokens received from the protocol share reserve. Any remaining balance in a converter contract after `updateAssetsState` is called will be available for performing incentivized conversions.

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

The access control manager or owner can perform the following privileged actions:

- Pause and resume conversions
- Add conversion configurations
- Add converter contracts to a converter network
- Set the following state variables:
  - The address of the price oracle in token converters
  - The destination address of converted funds in token converters
  - The address of the converter network in token converters
- Sweep (transfer out) any amount of arbitrary tokens from token converters

Medium Severity

M-01 Private Conversions Will Fail if Asset Includes Fee

During private conversions, `convertExactTokens` is used to convert an amount of the token being supplied by the calling converter to its base asset. However, the call will fail if the token being supplied includes a fee.

Consider performing conversions while also supporting tokens with fees on transfer.

Update: *Resolved. The Venus team stated:*

> *Currently, we are not providing support for fee-on-transfer tokens in private conversions. So, if there would be a token with fee-on-transfer in the system (we don't have any yet), the conversionAccess would be set to ONLY_FOR_USERS in the ConversionConfig associated*

Low Severity

L-01 Users Might End Up Paying More Tokens Than Expected During Conversion

The function `_convertForExactTokens()` is supposed to convert the specified exact output amount to an input amount, and then check that this input amount does not exceed the specified max input value. During this process, the function checks that `actualAmountIn` does not exceed `amountInMaxMantissa`. However, `actualAmountIn` is not the amount the user pays, it is the amount received by the converter after potential token transfer fees. This could cause the user to pay more than the max value they specified.

Consider checking that the `amountInMantissa` does not exceed the max value.

Update: *Resolved at commit 166d40b.*

L-02 Functions to Get AmountIn and AmountOut Might Not Be Accurate

Functions `_getAmountOut()` and `_getAmountIn()` in the `AbstractTokenConverter` contract are designed to calculate the conversion amount between the two tokens based on the current oracle prices and conversion incentives if there are any. However, these functions do not deal with potential token transfer fees. This could lead to inaccurate estimation outcomes and cause unexpected consequences (e.g., when using external view functions `getAmountIn` and `getAmountOut` to calculate expected trade sizes).

While it is difficult to get an accurate estimation of token transfer fees, consider clearly documenting that these functions do not account for potential token transfer fees.

Update: *Resolved at commit 6b6e9e4.*

L-03 Misconfigured Converter Network Could Cause Private Conversions to Pay Unnecessary Incentives

When adding a converter to a network, the system requires the converter to set its `converterNetwork` to the same network being updated. However, the owner can freely change a converter's network afterwards. If this converter is not manually removed from the old network, it will cause an uneven recognition between the converter and the network where the converter does not recognize the network, but the network recognizes the converter. This could lead to private conversions paying the incentive if a new network does not recognize the calling converter.

Consider adding a check in the `_findTokenConverters()` function to ensure that all returned converters are indeed recognizing the calling `converterNetwork` contract.

Update: *Resolved at commit 6bd7f9a.*

L-04 No Minimum Amount for Private Conversion

Private conversion will try to convert as much as possible from a given amount of tokens by looking through all possible converters in the network that have a balance. This is a computationally expensive operation that may not be worth the gas cost if these balances are very small.

Consider specifying a minimum acceptable balance to convert.

Update: *Resolved at commit f3e40c3, at commit 554d804.*

Notes & Additional Information

N-01 Inconsistent Use of Named Returns

There are multiple contracts throughout the codebase that have inconsistent usage of named returns in their functions:

- The `ConverterNetwork` contract in `ConverterNetwork.sol`
- The `RiskFundConverter` contract in `RiskFundConverter.sol`
- The `SingleTokenConverter` contract in `SingleTokenConverter.sol`

Consider using named returns consistently in all functions.

Update: *Resolved at commit 1af57c2.*

N-02 Lack of Indexed Event Parameters

Throughout the codebase, there are several events without any indexed parameters:

- The `ConvertedExactTokens` event in `AbstractTokenConverter.sol`
- The `ConvertedForExactTokens` event in `AbstractTokenConverter.sol`
- The `ConvertedExactTokensSupportingFeeOnTransferTokens` event in `AbstractTokenConverter.sol`
- The `ConvertedForExactTokensSupportingFeeOnTransferTokens` event in `AbstractTokenConverter.sol`
- The `AssetTransferredToDestination` event in `SingleTokenConverter.sol`

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

Update: *Resolved at commit b1244b3.*

N-03 Typographical Errors

An instance of a typographical error was identified in the codebase:

- At Line 630 of the `AbstractTokenConverter` contract, the comment has an extra `it` at the end.

Consider fixing this along with any other instances of typographical errors in the codebase.

Update: *Resolved at commit 9994f12.*

N-04 Repetitive Code in Functions

Functions `convertExactTokens` and `convertExactTokensSupportingFeeOnTransferTokens` in the `AbstractTokenConverter.sol` contract largely share the same code. So do functions `convertForExactTokens` and `convertForExactTokensSupportingFeeOnTransferTokens`.

Consider refactoring these functions to reduce code repetition.

Update: *Resolved at commit af3d4fc.*

N-05 Unnecessary Calculation and Reassignment

The function `getAmountIn` calculates `amountInMantissa`. However, this is unnecessary since the same calculation is obtained from the internal call in `_getAmountIn`. Consider removing this calculation in `getAmountIn` to reduce the extra code. In addition, the logic used to determine the `AmountConvertedMantissa` can be moved before the internal call, and can be used to replace `amountOutMantissa` as the argument to remove the entire `if` statement.

Update: *Resolved at commit ae9e07b.*

N-06 Functions Returning Parameters

In `_getAmountIn`, an unmodified function parameter value is assigned to a return variable, and likewise is done in `_getAmountOut`.

To remove extraneous code and improve the overall code clarity, consider removing these return values from the functions and modifying any additional logic as needed.

Update: *Resolved at commit 84cbf18.*

N-07 Unused Named Return Variable

The `balanceOf` function signature in the `AbstractTokenConverter` abstract contract contains a named return variable `tokenBalance`. However, this variable is not used in the definition of this function in the inheriting `RiskFundConverter` contract.

To improve the clarity of the codebase, consider keeping function signatures consistent between interfaces/abstract contracts and the contracts that implement them.

Update: *Resolved at commit a265d02.*

Client Reported

CR-01 Rounding in Conversions Exceeds Contract Balance

During private conversion, the computed amount of tokens to convert rounds up. If this amount in does not exceed the total amount to convert, it will be the amount used for conversion, thus requiring the total balance of base tokens from the converting contract. However, when the inverse calculation is done during the conversion, the amount out will exceed the converter's balance of base tokens due to the previous rounding up, causing a revert in the `_doTransferOut` function.

The client submitted a fix in commit a3bd0b8 of pull request #71.

We reviewed the reported issue and agree the described scenario is possible. We confirmed the fix implemented by the client removed rounding up to avoid exceeding the contract balance during private conversion. Rounding up remains as the behaviour during public conversions to avoid the potential risk of protocol insolvency.

Conclusion

The Venus protocol's revenue distribution system implements a protocol share reserve contract which allocates tokens to different converter contracts. These converters facilitate decentralized token conversions at discounted prices, thereby incentivizing users. The process, known as private conversion, optimizes conversions by matching complementary offerings from various converter contracts. After receiving funds, converters transfer base assets to their designated addresses. The protocol aims to streamline conversions while minimizing the risks associated with open-market token exchanges and large trade sizes.

The audit of the token converter and private conversions modules yielded one medium-severity issue and several low-severity issues. A number of recommendations aimed at improving the overall clarity and efficiency of the codebase were also made. The Venus team was highly communicative throughout the audit process and answered any questions we had about the system.