Generic Money

Security Review

Cantina Managed review by:
Joran Honig, Lead Security Researcher
Wei Tang, Lead Security Researcher

January 19, 2026

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings are a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Generic provides stablecoin-as-a-service that delivers you the best risk-adjusted onchain yield and payments-ready privacy without the overhead and cost of offchain issuers.

From Nov 3rd to Nov 17th the Cantina team conducted a review of generic-protocol and generic-bridging on commit hashes 67cb0291 and 7a6c887b respectively. The team identified a total of **8** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	2	0	2
Medium Risk	2	2	0
Low Risk	1	1	0
Gas Optimizations	1	1	0
Informational	2	0	2
Total	**8**	**5**	**3**

2.1 Scope

The security review had the following components in scope:

```
├── generic-bridging
│   └── src
│       ├── adapters
│       │   ├── BaseAdapter.sol
│       │   ├── BridgeTypes.sol
│       │   ├── LayerZeroAdapter.sol
│       │   └── LineaBridgeAdapter.sol
│       ├── BridgeCoordinatorL1.sol
│       ├── BridgeCoordinatorL2.sol
│       ├── coordinator
│       │   ├── AdapterManager.sol
│       │   ├── BaseBridgeCoordinator.sol
│       │   ├── BridgeCoordinator.sol
│       │   ├── BridgeMessageCoordinator.sol
│       │   ├── EmergencyManager.sol
│       │   ├── Message.sol
│       │   └── PredepositCoordinator.sol
│       └── utils
│           └── Bytes32AddressLib.sol
└── generic-protocol
    └── src
        ├── controller
        │   ├── AccountingLogic.sol
        │   ├── BaseController.sol
        │   ├── ConfigManager.sol
        │   ├── Controller.sol
        │   ├── EmergencyManager.sol
        │   ├── PeripheryManager.sol
        │   ├── PriceFeedManager.sol
        │   ├── RebalancingManager.sol
        │   ├── RewardsManager.sol
        │   ├── VaultLimitsLogic.sol
        │   ├── VaultManager.sol
        │   └── YieldManager.sol
        ├── GenericUSD.sol
        ├── periphery
```

```
│   ├── GenericDepositor.sol
│   └── swapper
│       ├── OneInchSwapper.sol
│       ├── UniswapV3Swapper.sol
│       └── USDSToDAIUniswapV3Swapper.sol
├── unit
│   ├── ERC20Mintable.sol
│   ├── GenericUnit.sol
│   ├── GenericUnitL2.sol
│   └── whitelabeled
│       └── WhitelabeledUnitUpgradeable.sol
├── utils
│   └── tryGetAssetDecimals.sol
└── vault
    ├── ControlledERC7575Vault.sol
    ├── GenericVault.sol
    └── SingleStrategyVault.sol
```

2.2 Limitations of peg maintenance mechanism

Derived PEG maintenance: The generic money stablecoin peg maintenance is interesting as its peg maintenance mechanics rely on the peg maintenance behaviour of collateral tokens.

One of the results of this design is that the system does not always have incentives in place that push the price of the stablecoin to it's peg. This happens in the following two cases:

- If all collateral tokens are above their peg value then GUSD will have no mechanism to move its price down beyond the least valuable collateral token.
- If all collateral tokens are below their peg then GUSD will follow the price of the most valuable collateral token.

This mechanism relies on the underlying stablecoins to reacquire their peg and is not able to achieve this on its own. To balance this deficit the peg maintenance benefits from the ”strongest link in the chain” (up to a limit based on collateral allocation).

Bankrun Resistance: An important scenario that stable coins need to deal with is bankruns.

A bankrun is a situation in which a lot of GUSD would be redeemed in a short amount of time. It's important to ensure there is sufficient liquidity to ensure redemptions can continue functioning correctly. To achieve this many protocols often implement mechanics to unwind underlying collateral positions.

The generic money protocol puts most of the available collateral in lending pools, which also include mechanics to ensure sufficient liquidity for depositor withdrawals, often through implementing a fee curve based on utilisation.

However, the liquidity requirements of lending pools are less strenuous than those of a stable coin. This is the case because the peg maintenance mechanisms often rely on facilitating redemptions. As a result, the ability of GUSD to maintain its peg in a bankrun scenario is limited since the protocol can potentially be limited by the rate at which underlying vaults allow withdrawal.

Careful configuration of the safety buffer and vault strategies can allow the operator of the generic money protocol to minimize the risk of this scenario occurring.

PEG maintenance and expected future value: The goal of a stable coin protocol is to introduce a set of incentives that ensure that the expected value of a token is equal to the value of some predetermined peg (such as the US dollar). A less obvious goal is that a peg maintenance mechanism should maintain not just the expected value at t_{now}, but also the expected value of the token for $t_{\text{now}+x}$ for any x at t_{now}. Not just that, you want to minimise the expected error (deviation from the peg). In other words, you expect that a stable coin maintains its value over time.

It's useful to formulate some assumptions to explore how the present design influences these expected values:

- There is no risks of hacks of the generic protocol or the underlying vaults (this assumption, while invalid, makes reasoning about the mechanisms much easier).

- The expected value of collateral stable coins eventually converges on their peg and as such the expected value of one collateral token is equal to that of any other collateral token at $t_{\text{now}+x}$ for any sufficiently large x.
- There is a non-zero risk in deploying capital in vaults.
- The expected yield is and higher than the risk incurred by deploying capital in a vault.

An important aspect of the generic money design is that the expected yield is allocated differently than the expected losses from the vaults. The protocol first allocates yield to maintain a relatively small margin, but then awards the yield to a third party. However, the potential loss/debt incurred by the vaults is socialised amongst the GUSD holders.

From the assumptions and the mechanics it must follow that expected value of the generic money stable coin at $t_{\text{now}+x}$ for any sufficiently large x must be lower than the expected value of any of the collateral stables. This is the case because while GUSD holders have a non-zero expectation of loss they do not have an expected profit (with respect to the PEG).

Note that this does not imply that all the profit must be awarded to GUSD token holders. Instead it's recommended to leverage (part of) the profits in a mechanism which minimises the expected deviation from peg due to risk incurred by the positions held by the protocol. Controls such as diversification of protocol held positions (including non-yield bearing stable coin holdings) and configuration of the safety buffer are both relevant in achieving the goal of minimizing the expected error.

3 Findings

3.1 High Risk

3.1.1 Insufficient slippage control for vault operations

Severity: High Risk

Context: RebalancingManager.sol#L99-L111, RebalancingManager.sol#L128-L161

Description: Various aspects of the protocol effect a change in the amount of assets deployed in a strategy without accounting for slippage.

The ERC4626 standard, while often used with an assumption of monotonically increasing share prices, does not guarantee that this is the case. A good example would be a vault/ strategy that deposits assets in Curve's 3pool. In such cases the protocol should account for the fact that operations that change the allocated collateral are potentially vulnerable to slippage.

This affects both user operations (deposits and withdrawals) and privileged operations such as those used to rebalance assets between vaults.

Recommendation: The simplest option is to restrict the underlying strategies such that we can assume there would only be monotonically increasing share prices. The design space for completely autonomous implementation of deposits and withdrawals from a multi-vault protocol is large, and suggesting any particular solution is out of scope for this audit.

Generic Money: Acknowledged. Protocol governance is responsible for managing available strategies and will restrict them to only those with monotonically increasing share prices.

Cantina Managed: Acknowledged.

3.1.2 Vault price manipulation allows yield manager to mint tokens

Severity: High Risk

Context: YieldManager.sol#L51-L61

Description: The yield manager can mint additional tokens whenever the value of the backing assets is higher than the total amount of shares.

The method for determining the current value of backing assets relies on a call to the underlying vault `_additionalOwnedAssets()` functions. However, the return value of this function is potentially manipulable. Issue #1 describes a scenario which leverages this same property.

Take for example a vault which deposits in Curve 3pool. In normal cases `_additionalOwnedAssets()` returns the correct value of the LP position that the vault holds. However, if an attacker manipulates the pool by selling a lot of the vault asset (increasing its balance in the pool) then the generic protocol will assume the vault's LP position is much more valuable.

If an attacker is able to manipulate the perceived amount of backing assets then they're also able to make the protocol (temporarily) think it has more yield than the safety buffer. As a result anyone could sandwich a `distributeYield` call to make the system erroneously mint shares.

This includes the periphery manager who could front run and update the yield distributor address and in so doing steal funds from the beyond the excess yield.

Recommendation: One solution to this problem is to limit the vaults/strategies such that vaults have monotonically increasing share prices. Alternatively you might leverage an oracle to get a true value of the surplus yield.

Generic Money: Acknowledged. The Yield Manager is a trusted role in the protocol. We limit the strategies to lending markets that satisfy the requirement of monotonically increasing share prices, though this is enforced via governance rather than in the code.

Callout in docs Diversification.

Cantina Managed: Acknowledged.

3.2 Medium Risk

3.2.1 Periphery manager can manipulate rebalance and rewards and extract funds

Severity: Medium Risk

Context: RebalancingManager.sol#L128-L160, RewardsManager.sol#L69-L89

Description: The periphery manager is a semi-trusted actor that's allowed to control the yield distributor and swapper contract addresses. Though it's expected that this actor can steal some funds from the protocol they should not be able to steal more funds than the surplus yield.

However, the rebalancing logic and rewards claim functionality relies on a truthful return value to the `_swapper.swap()` call to check slippage. Since this value is indirectly controlled by the periphery manager, they would be able to have the swap incur more slippage than allowed. Similarly, the check does not inhibit the rebalancing manager from incurring slippage as they freely control the `minAmountOut` value.

There are two additional checks in the rebalancing logic that limit the extent to which the periphery manager can profit from such an attack:

1. A check that ensures the total underlying value does not decrease by too much.
2. A check that ensures the safety buffer is maintained.

There is an opportunity for both the periphery manager and the rebalancing manager actors to circumvent the first of these two checks. Note that the check ensures the total value does change, not the relative value of each share. This creates a re-entrancy opportunity for the external call that occurs to make the swap. A malicious actor that controls (part of) the external call can deposit funds in the protocol during this external call and increase the backingAssetsValue with the amount of assets that they just extracted, only to withdraw most of those assets after the attack has been executed. Since the periphery manager controls the external call measure (1) is ineffective. It depends on the swapper implementation whether the rebalancing manager is able to control execution at some point during the external call.

The second additional measure is sometimes disabled as rebalancing assets is necessary even when the safety buffer is depleted. This is not unexpected, so a periphery manager can wait for this to happen. As a result measure (2) is only partially effective.

Additionally, as also mentioned in issues #1 and #2, the backing assets value is potentially manipulable depending on the vault implementations. This also affects the efficacy of the additional slippage checks that might otherwise prevent exploitation by the periphery manager.

In summary, it is likely that there are multiple points in time where the periphery manager can steal a portion of the funds in the protocol.

Recommendation: We recommend two defensive changes to the code base:

1. Do not trust the return value of the swapper contract and instead leverage the return value of a balanceOf call to the collateral asset before and after the swap. This same measure can also be applied to harden the swapper contracts themselves.
2. Do not allow the total amount of shares to change during the rebalancing operation, while also keeping the existing slippage check on total backing assets value in place.

Addressing the manipulable of the perceived backing assets value is discussed in issues #1 and #2.

Generic Money: Fixed in PR 180.

Cantina Managed: Fix verified.

3.2.2 An attacker can (temporarily) freeze deposits and withdrawals

Severity: Medium Risk

Context: *(No context files were provided by the reviewer)*

Description: An attacker can (temporarily) freeze the funds in all vaults by cleverly manipulating the vault balances and triggering a kind of deadlock in the vault limit mechanisms.

In normal operations we can expect to encounter a situation where a single vault has hit their min proportionality. This causes deposits to all other vaults to be blocked until someone deposits in the desired vault or withdraws from another vault. Similarly a vault that has hit their max proportionality with prevent withdrawals from other vaults (and a deposit in the given vault). A problem arises when two vaults simultaneously hit their limits. If two vaults hit their max proportionality limit then withdrawals are completely disabled and the only method of recovery would be a deposit to another vault.

An attacker can (temporarily) block deposits and withdrawals by leveraging this effect. An example with four vaults:

- Vault 1 + 2: Withdraw until min proportionality metric is hit (you can not deposit in any vault).
- Vault 3 + 4: Deposit/ donate until over max proportionality metric (you can not withdraw from any vault).

Limitations: The donation which enables the attack also enables anyone to resolve the problem by simply donating until the min proportionality of vault 1 & 2 is held. At this point you can deposit in these two vaults until max proportionality of 3 + 4 is also satisfied.

In addition, the rebalancing manager can also rebalance funds to unfreeze the assets.

Additional Notes: The rebalancing mechanism does not check the vault proportionality limits. As a result an attacker might sandwich the rebalancing transaction in such a way that it puts the vaults in the deadlock state. In such a scenario the attacker does not have to loose money for donations.

This also makes it possible to put the system in a state where immediate recovery through the method described above becomes prohibitively expensive.

Finally, the rebalancing manager and config manager can also intentionally put the system in a locked state preventing withdrawals and deposits.

Recommendation: The design space for a solution is too broad to provide an immediate recommendation. However, the following are high level observations on a potential remediation strategy. A simple approach is to consider the composition of proportionality limits such that limits can not be hit simultaneously. For example if you ensure that any possible subset of vaults of size 2 or higher has a cumulative proportionality larger than 1 then you're sure that withdrawals can not be locked due to a deposit lock.

Alternatively one might consider a proportionality approach based on minimising the error (difference from proportionality requirement) instead of an absolute check whether there is a zero error.

Generic Money: Fixed in PR 179.

We now check vault proportional limits after adding, removing, and updating vaults settings. The rules are as follow:

- 1 vault: max proportionality must be equal to 100%.
- 2 vaults: sum of max proportionalities must be greater than or equal to 120%.
- 3 and more vaults: sum of two smallest max proportionalities must be greater than or equal to 100%.

Cantina Managed: Fix verified. The pull request fixes the issue.

There is some rounding happening in the max delta calculations so you could consider requiring a slightly smaller percentage (e.g. 101%). However, I'm not sure that this is strictly necessary, as for this to be exploitable there would need to be another vault at their min proportionality limit. This would need to be some value very close to but not exactly zero, which I assume is not an expected/ realistic configuration.

3.3 Low Risk

3.3.1 Message id's are not strictly unique which potentially prevents rollbacks

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: The message id is determined based on the target chain id, the bridge type, the block timestamp at creation of the message and the nonce of the adapter creating the message. It's possible that two adapters on different chains create a message for the same target chain, with the same bridge type, at the same timestamp with the same nonce. Message IDs should be unique and the result of this collision affects rollbacks for example. The rollback bookkeeping assumes each message has a unique ID and will allow the last registered failed message to be rolled back. This is because the second message failing essentially overwrites the record of the first message failing.

Recommendation: Including the source chain ID in calculating the message ID should resolve this issue.

Generic Money: Fixed in PR 8.

Cantina Managed: Fix verified. The pull request fixes the issue by including the source id in the message id computation.

3.4 Gas Optimization

3.4.1 Use `uint256` for nonce will result in shorter assembly

Severity: Gas Optimization

Context: BaseAdapter.sol#L82-L84

Description: The `BaseAdapter` in Generic Bridging defines a `nonce` storage value. The `nonce` is used for hashing in the generation of message ID, and is incremented on every bridge message. Currently, `nonce` is defined as `uint32`. It is assumed that there will practically never be possible to have more than `uint32::MAX` messages.

Switching to the full length `uint256` will actually result in less code generation and therefore gas savings. A storage access requires at least a full-length (256-bit) operation. In addition, the compiler is not required to carry out other checks to make sure that the value stay in the range of `0..uint32::MAX`.

We use Solidity 0.8.30 to check the generated code. `unchecked { ++nonce; }` when `nonce` is `uint32` resulted in the following EVM opcodes:

```
PUSH 0
PUSH 0
DUP2
DUP2
SWAP1
SLOAD
SWAP1
PUSH 100
EXP
SWAP1
DIV
PUSH FFFFFFFF
AND
PUSH 1
ADD
SWAP2
SWAP1
PUSH 100
EXP
DUP2
SLOAD
DUP2
PUSH FFFFFFFF
MUL
NOT
AND
SWAP1
DUP4
PUSH FFFFFFFF
AND
MUL
OR
SWAP1
SSTORE
POP
```

`unchecked { ++nonce; }` when `nonce` is `uint256` resulted in the following EVM opcodes:

```
PUSH 0
PUSH 0
```

```
DUP2
SLOAD
PUSH 1
ADD
SWAP2
SWAP1
POP
DUP2
SWAP1
SSTORE
POP
```

The operation always begins with a full-length `SLOAD`. If `nonce` is `uint256`, the compiler would then immediately add 1 and call `SSTORE`. On the other hand, if `nonce` is `uint32`, several additional operations are needed to make sure the value stays in range.

Recommendation: Switch `nonce` declaration to `uint256`. It may optionally be converted to a smaller integer type in message ID generation if necessary.

Note that switching to `uint256` will result in using an additional storage slot in Solidity's storage layout. However, in the case of this particular contract, the other storage value that is being packed with is `_pendingOwner` from `Ownable2Step`, which is seldom used, and never read/write together with the nonce. Therefore, simply switching to `uint256` will be *strictly faster* than any shorter integer types. However, if the additional storage slot is actually an concern, then switching to at least `uint64` as in finding#8 is at least recommended.

Generic Money: Fixed in PR 8: updated `nonce` type to `uint256`. PR 10 update `nonce` to `uint64` after refactoring `BridgeCoordinator` due to other findings.

Cantina Managed: Fix verified.

3.5 Informational

3.5.1 32-bit nonce can run out

Severity: Informational

Context: BaseAdapter.sol#L30-L33

Description: The bridge adapter defines a 32-bit nonce value. Over a long, but perceivable period, the nonce value can run out.

- At this moment, the address with the highest nonce on Ethereum is `0xEA674fdDe714fd979de3EdF0F56AA9716B898ec8` with over 48 million transactions in 10 years. To reach the maximum of 32-bit (~4294 million), it will take 10 times of that.
- However, under potential attack scenario and with gradually improved network throughput of Ethereum, we should expect it to run out much faster. Currently, daily used gas on Ethereum is around 163 billion. If all of those gas were to the bridge adapter, and assuming the cheapest transaction of 21000 gas, with `2^32 - 1` transactions, the nonce will run out in only `21000 * (2^32 - 1) / 163000000000 ~= 554` days.

Recommendation: Switch the `nonce` definition to `uint64` or `uint256`. 64-bit still fits the contract's storage layout for storage packing, and this is also the same definition as in EIP-2681. `uint256` has slight advantage in gas optimization, but it requires one additional storage slot.

Generic Money: Fixed in PR 10.

Cantina Managed: Fix verified.

3.5.2 Missing vault update events

Severity: Informational

Context: GenericUnit.sol#L51-L52

Description/Recommendation: According to ERC-7575 the implementation: SHOULD emit the `VaultUpdate` event when a Vault linked to the share changes. Note that this is not a MUST requirement and therefore inclusion is discretionary.

Generic Money: Acknowledged.

Cantina Managed: Acknowledged.