OpenZeppelin | security

Mountain Protocol USDM Audit

Audit project logo

December 21, 2023

Table of Contents

Summary

Type	DeFi
Timeline	From 2023-05-31 To 2023-06-02
Languages	Solidity

Total Issues	10 (8 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	1 (1 resolved)
Low Severity Issues	2 (1 resolved)
Notes & Additional Information	7 (6 resolved)
Client Reported Issues	0 (0 resolved)

Scope

We audited the mountainprotocol/tokens repository at the d548aa6f037d9c7ed653a8004f83949b70133c7a commit.

The fixes to our findings were finalised in the same repository at the 2f0e2f08362fc44360357627b390d21b42031265 commit. After the review, the `USDM` token has been deployed with this latest mentioned commit at address `0x59D9356E565Ab3A36dD77763Fc0d87fEaf85508C`. Notice that the mentioned address is a proxy contract that, at the time of writing, points to the version in the finalised commit. Because of this, the proxy can be upgraded to point to a new implementation, different from the current and reviewed one. The token address is also the same across all supported chains.

In scope were the following contracts:

```
contracts
└── USDM.sol
```

System Overview

USDM is a stablecoin, intended to be valued at 1 USD, backed by short-term US Treasury bills. The Mountain Protocol team will retain a small fraction of the corresponding yield, with the rest distributed to USDM token holders through rebasing. In this way, the price of USDM should remain stable while token holders see their balance of USDM increase daily.

The Mountain Protocol hosts the primary market that establishes the price peg. In particular, registered customers can deposit collateral (either USDC tokens or fiat currency) in exchange for USDM, redeem USDM for collateral, or withdraw USDM to any Ethereum address. Since they are offered an exchange rate of one dollar per USDM in both directions, any price movement in the external (secondary) market can be arbitraged in the primary market to re-establish the peg.

The secondary market includes any external system that deals with USDM. The Mountain Protocol intends to establish a USDC-USDM Curve Pool on Ethereum.

Rebasing considerations

The Mountain Protocol team intends to rebase USDM daily to account for the investment yield. This means that during normal operation, the USDM balance of all token holders should steadily increase. To avoid potential loss of funds, users should ensure that any smart contracts that receive USDM tokens are capable of handling changing balances.

Security Model and Trust Assumptions

The primary market is run by the Mountain Protocol team, and the secondary market also depends on its operation. Therefore, users must implicitly trust the team to maintain the availability of the market, invest the collateral safely and rebase USDM in line with the yields.

Users also implicitly assume that short-term US Treasury bills continue functioning as a safe investment.

Lastly, there are several privileged roles in the system, all managed by the Mountain Protocol team:

- Anyone with the Upgrade role can arbitrarily replace the USDM token contract, which effectively gives them complete control over all balances.
- Anyone with the Minter role can create and assign new USDM tokens. This should be used to increase the USDM supply to account for net daily collateral deposits in the primary market.
- Anyone with the Burner role can burn USDM tokens from any address. This should be used to decrease the USDM supply to account for net daily collateral redemptions in the primary market. Tokens should only be burned from the Mountain Protocol team's own hot wallet.
- Anyone with the Blocklist role can manage the blocklist, which contains addresses that cannot transfer USDM tokens (although they can still receive them). Interestingly, this also means that if they block the zero address, all minting will be disabled as well. Finally, blocked users cannot have their tokens burned.
- Anyone with the Pause role can pause all token transfers, including minting and burning.
- Anyone with the Oracle role can rebase USDM to increase the balance held by all users. This is expected to occur daily.
- Anyone with the Administrator role can rebase USDM in either direction to increase or decrease the balance held by all users. This should never occur during normal operations. In addition, they cannot decrease the USDM token supply to less than the supply of shares.
- The Administrator role also manages all other roles, which means the administrator can choose which addresses have which powers.

Medium Severity

M-01 Unclear the Blocklist Limitations

The blocklist feature is meant to prevent some users from transferring `USDM` tokens. The list of blocked addresses is managed by any address with the `BLOCKLIST_ROLE` role.

All token transfers only ensure that the source of the tokens is not blocked. This has several implications:

- blocked addresses can receive USDM tokens through the `mint` and `transfer`/`transferFrom` functions, but they cannot transfer them out.
- addresses on the blocklist cannot have their tokens burned.
- a blocked address still can spend a token allowance or grant allowances to other addresses, although the grantee will still not be able to `transferFrom` the blocked address.

Consider whether this is the expected behavior in all four functionalities (`mint`, `transfer`, `burn` and approvals) and whether the code should be updated accordingly. Moreover, consider explicitly documenting the behavior in all functions' docstrings.

Update: *Resolved in commit 92584df. The Mountain Protocol confirmed that this is the desired behavior and expanded the smart contract documentation accordingly. They also stated:*

> *In an effort to avoid increased gas fees on all users, a verification of destination blocked address has purposefully not been set in place. Users are expected to know parties they are transacting with and avoiding blocked addresses.*

Low Severity

L-01 Discontinuous Reward Pattern

Rebases occur by discontinuously increasing all user balances. This implies that a USDM token purchased immediately before the rebase transaction is worth more than one purchased

immediately afterwards. In principle, this means that the price of USDM tokens should follow a sawtooth wave.

In practice, the effect size will be minimal because the daily interest rate is expected to be less than 0.02%. Nevertheless, we believe it is worth considering because:

- The expected impact depends on external factors, such as the actual yield, the availability of Ethereum, the reliability and availability of Mountain Protocol's external infrastructure, any events that require pausing the contract, etc.
- Any discontinuity creates front-running, back-running or transaction delay opportunities. In this case, a transfer of a fixed number of USDM is worth slightly less if executed after the update. Similarly, anyone who wants to spend an allowance would get more value if they spent it before the update. Even with a small effect size, it could introduce a new market behavior.

Consider whether it is worth the extra complexity to smooth the discontinuity. This would involve distributing the daily yield over time rather than in one step.

Update: *Acknowledged, not resolved. The Mountain Protocol team stated:*

> *Given the minimal effect size, smoothing the rebases is not worth the extra complexity. However, we will describe the edge case in our risk documentation.*

L-02 Misleading Comments

There are some places in the codebase where comments are either inaccurate or misleading.

- The `addRewardMultiplier` has a comment that says that the input parameter `_rewardMultiplier` is the new `rewardMultiplier` value, but this is not true since it is actually an increment on top of it.
- The `setRewardMultiplier` comment says it can be called by `ADMIN_ROLE` but it is actually `DEFAULT_ADMIN_ROLE`.
- The comment above the `_mint` function says "Internal function" but it should be "Private function" instead.

To improve correctness and readability, consider fixing these examples and reviewing the codebase in case other comments can benefit from rephrasing.

Update: *Resolved in commit fdaf392.*

Notes & Additional Information

N-01 Dust Can Accumulate

To accommodate rebasing, the user-facing functions describe token amounts while the contract's own logic uses the equivalent amount of shares. Both conversion functions (`convertToShares` and `convertToAmount`) round down, which could lead to minor discrepancies.

For example, `0.1e18` USDM tokens at a `rewardMultiplier` of `1.2e18` will convert to `8333333333333333` shares, which will convert back to `99999999999999999` USDM tokens.

In the interest of safety, all rounding errors should be in favor of the protocol, and the contract correctly implements this heuristic. Nevertheless, the consequence is that all token transfers (including mint and burn operations) may transfer slightly less than expected. If a user attempted to transfer their entire balance, they could retain some shares in their wallet, corresponding to a fraction of a token.

Consider documenting this behavior in the docstrings so that users can be aware of it. In addition, consider whether a `transferShares` function (or possible `transferAll` function) should be included so that knowledgeable users can have finer control.

Update: *Resolved in commit 4606704.*

N-02 Imprecise Reward Multiplier Restriction

The `_setRewardMultiplier` function of the `USDM` contract ensures the parameter is not less than `1e18`. This correctly represents 100%, since it is the same numeric value as the multiplier precision constant. Nevertheless, in the interest of predictability and local reasoning, consider using the existing constant in the validation.

Update: *Resolved in commit 2e9b94c.*

N-03 Missing Interface

The `USDM` contract implements ERC-20 metadata functions but does not inherit the corresponding interface. Consider explicitly inheriting relevant interfaces to ensure consistency and to document the intended behavior.

Update: *Resolved in commit a532c49.*

N-04 Naming Suggestions

There are some names used throughout the codebase that might benefit from a change:

- The `convertToAmount` function should be `convertToToken`. Both conversion functions use the term "amount" to describe both shares and tokens and it might be confusing.
- The `_rewardMultiplier` input parameter of the `addRewardMultiplier` function should be something like `_rewardMultiplierIncrement` instead.
- While the `_blocklist` mapping has an accurate name since it represents a list of blocked accounts, the `blocklistAccounts` and `unblocklistAccounts` functions might be just `blockAccounts` and `unblockAccounts`, since `blocklist` is not a verb. The same applies to the `AccountBlocklisted` and `AccountUnblocklisted` events.

Consider adopting the suggested naming conventions to improve the readability and clarity of the codebase.

Update: *Resolved in commit 500a2c9.*

N-05 Reuse OpenZeppelin Implementation

The `USDM` contract reimplements a lot of functionality that is already available in the OpenZeppelin `ERC20PermitUpgradeable` contract with minor modifications to account for the rebasing logic. However, it would be simpler to inherit from the contract and introduce the modifications by overriding the relevant functions. Specifically, our suggestion is:

- Treat the existing _balances and _totalSupply as the `_shares` mapping and `_totalShares` variable respectively. Since these are private values, the original names never need to be shown to the user.
- Update the balance getter functions accordingly.

- Update the `_mint`, `_burn` and `_transfer` functions to simply override and invoke the OpenZeppelin version, first converting the `amount` parameter to the equivalent number of shares.
- Remove the obsolete variables and reimplemented code. This includes the metadata getters, `transfer` function and bottom third of the contract.

In this way, the `USDM` contract would only focus on the new functionality that it introduces.

Update: *Acknowledged, not resolved. The Mountain Protocol team stated:*

> *In the interest of code clarity and avoiding confusion, we do not want to record shares in the `_balances` mapping.*

N-06 Unexpected Storage Gap Size

The `USDM` contract contains a storage gap of 50 storage slots to account for potential future changes to the storage layout. However, the convention (as noted in the referenced document) is to choose the size so the contract's variables, including the `__gap`, use a combined 50 storage slots. In this way, future versions of the contract will have a computable gap without reference to the original version.

Consider changing the size of the variable to correctly follow the convention.

Update: *Resolved in commit de984d1.*

N-07 Use of Deprecated Function

The `USDM` contract uses the `_setupRole` function, but this function has been deprecated. Consider using `_grantRole` instead.

Update: *Resolved in commit [45a85a3](#).*

Conclusions

A medium-severity and other lower-severity issues have been found. Several fixes were suggested to improve the readability (or clarity) of the codebase and facilitate future audits, integrations and development. The Mountain Protocol team was very responsive and provided extensive documentation about the project.

Below we provide our recommendations for monitoring important activities in order to detect and prevent potential bugs.

Appendix

N-01 Monitoring Recommendations

While audits help in identifying code-level issues in the current implementation and potentially the code deployed in production, the Mountain Protocol team is encouraged to consider incorporating monitoring activities in the production environment. Ongoing monitoring of deployed contracts helps identify potential threats and issues affecting production environments. With the goal of providing a complete security assessment, the monitoring recommendations section raises several actions addressing trust assumptions and out-of-scope components that can benefit from on-chain monitoring.

It is recommended to monitor all sensitive actions that affect the token contract, including:

- All calls to the `mint` and `burn` functions to ensure they are expected, match the daily net deposit/withdrawal amounts, and are within a reasonable range.
- All calls to the `blocklistAccounts` and `unblocklistAccounts` functions to ensure they are expected.
- All calls to the `addRewardMultiplier` and `setRewardMultiplier` functions to ensure they are expected and within a reasonable range.
- Whenever the protocol is paused or unpaused, to ensure it was expected.
- Whenever the contract is upgraded, to ensure it was expected and executed correctly.
- Any change in the role architecture, including when role holders are added or removed.

It is also recommended to monitor the secondary markets, including the health of the Curve USDC-USDM pool, to ensure:

- The USDM price remains sufficiently stable.
- The secondary market has sufficient liquidity.
- Any large changes in volume or price are identified and explained.