Midas Max Supply Update Audit

Côme du Crest

2025-08-30

Table of contents

Midas Max Supply Update Audit

This document presents the finding of a smart contract audit conducted by Côme du Crest for Midas Protocol.

Scope

The scope includes pull requests `80` of `midas-apps/contracts`. The pull request was first reviewed at commit `0xe508039`.

The reported issue has been fixed and the last commit reviewed of the pull request is `0xba85967`.

Context

This audit is the fourth audit I perform of this contracts codebase.

The following context was given for this audit:

> The deposit vault has a maxSupplyCap parameter (in 18 decimals), which can be adjusted by the vault admin.
>
> The minting transaction fails if the supply after the transaction falls above the maxSupplyCap parameter
>
> instant or standard modes are both falling into this cap, meaning depositInstant or any approve requests (batched, safe, or not) are all impacted
>
> safeBulk can still mint until the cap is reached
>
> Known Edge cases
>
> - The dv is only aware of itself:
>   - The cap is therefore within a network/chain
>   - other dvs of the same token may have a different cap
> - When not required, we will operationally set a very high cap, so mintings can always happen.

Status

The reported issue has been fixed and the fix reviewed and discussed directly with the developer.

No issue remain.

Legal Information And Disclaimer

1. This report is based solely on the information provided by Midas Protocol Ltd. (the “Company”), with the assumption that the information provided is authentic, accurate, complete, and not misleading as of the date of this report. The auditor has not conducted any independent enquiries, investigations or due diligence in respect of the Company, its business or its operations.
2. Changes to the information contained in the documents, repositories and any other materials referenced in this report might affect or change the analysis and conclusions presented. The auditor is not responsible for monitoring, nor will we be aware of, any future additions, modifications, or deletions to the audited code. As such, the auditor does not assume any responsibility to update any information, content or data contained in this report following the date of its publication.
3. This report does not address, nor should it be interpreted as addressing, any regulatory, tax or legal matters, including but not limited to: tax treatment, tax consequences, levies, duties, data privacy, data protection laws, issues relating to the licensing of information technology, intellectual property, money laundering and countering the financing of terrorism, or any other legal restrictions or prohibitions. The auditor disclaims any liability for omissions or errors in the findings or conclusions presented in this report.
4. The views expressed in this report are solely my views regarding the specific issues discussed within this report. This report is not intended to be exhaustive, nor should it be construed as an assurance, guarantee or warranty that the code is free from bugs, vulnerabilities, defects or deficiencies. Different use cases may carry different risks, and integration with third-party applications may introduce additional risks.
5. This report is provided for informational purposes only and should not be used as the basis for making investment or financial decisions. This report does not constitute investment research and should not be viewed as an invitation, recommendation, solicitation or offer to subscribe for or purchase any securities, investments, products or services. The auditor is not a financial advisor, and this report does not constitute financial or investment advice.
6. The statements in this report should be considered as a whole, and no individual statement should be extracted or referenced independently.
7. To the fullest extent permitted by applicable laws, the auditor disclaims any and all other liability, whether in contract, tort, or otherwise, that may arise from this report or the use thereof.

Issues

[High] Storage layout incompatibility

Summary

The new version of `DepositVault` adds a storage variable for `maxSupplyCap` in the middle of its storage layout breaking compatibility with already deployed versions of deposit vaults. Moreover, the initializer function has been updated to have an additional `maxSupplyCap` argument to set the max supply cap but pre-existing contracts will not have the max supply cap set which makes it harder to follow the state of the contracts.

Vulnerability Detail

The storage variable `maxSupplyCap` has been added in the middle of the storage of the contract and the `__gap` storage variable has not been updated to maintain storage size:

```
contract DepositVault is ManageableVault, IDepositVault {
    ...

    uint256 public minMTokenAmountForFirstDeposit;

    /**
     * @notice max supply cap value in mToken
     * @dev if after the deposit, mToken.totalSupply() > maxSupplyCap,
     * the tx will be reverted
     */
    uint256 public maxSupplyCap;

    /**
     * @notice mapping, requestId => request data
     */
    mapping(uint256 => Request) public mintRequests;

    /**
     * @dev depositor address => amount minted
     */
    mapping(address => uint256) public totalMinted;

    /**
     * @dev leaving a storage gap for futures updates
     */
    uint256[50] private __gap;

    ...
}
```

Moreover, the function `initialize()` has been updated to set the `_maxSupplyCap` value:

```
    function initialize(
        address _ac,
        MTokenInitParams calldata _mTokenInitParams,
        ReceiversInitParams calldata _receiversInitParams,
        InstantInitParams calldata _instantInitParams,
        address _sanctionsList,
        uint256 _variationTolerance,
        uint256 _minAmount,
        uint256 _minMTokenAmountForFirstDeposit,
+       uint256 _maxSupplyCap
    ) external initializer {
        __DepositVault_init(
            _ac,
            _mTokenInitParams,
            _receiversInitParams,
            _instantInitParams,
            _sanctionsList,
            _variationTolerance,
            _minAmount,
~           _minMTokenAmountForFirstDeposit,
+           _maxSupplyCap
        );
    }
```

Impact

If the existing contracts are updated to this new version, the storage variables `mintRequests`, and `totalMinted` will be broken. Additionally, the

Code Snippets

https://github.com/midas-apps/contracts/blob/e50803996f75b80dbdc4d57d7f3216c1f9570c81/contracts/DepositVault.sol#L85

https://github.com/midas-apps/contracts/blob/e50803996f75b80dbdc4d57d7f3216c1f9570c81/contracts/DepositVault.sol#L122

Recommendation

Move the `maxSupplyCap` declaration to just before the `__gap` variable and lower the size of the gap by 1 to keep storage layout consistent.

Have different initializer functions for the different contract versions to make it obvious what has been called on different existing contract versions. Have `reinitializer` functions with the `reinitializer(version)` modifier and have the `initialize()` function (with no initialize modifier) call sequentially the different versions of the init functions, the first one having the `initializer` modifier and the following ones `reinitialize(2)`, `reinitializer(3)` and so on.

It would look like this:

```
contract DepositVault {

    function initialize(...) public {
        initializeV1(...);
        initializeV2(...);
    }

    function initializeV1(...) public initializer {
        ...
    }

    function initializeV2(...) public reinitializer(2) {
        ...
    }
}
```

Response

The code has been updated in commit `0x779ff69` to provide two different initializers as suggested. The `maxSupplyCap` declaration has been moved and the gap resized as suggested.

The `__DepositVault_init()` internal function remains. It initializes the vault only up to version 1 and is unused so it could be removed.

The function `__DepositVault_init()` has been removed in commit `0xba85967`.

Optimisations and miscellaneous

This part lists minor gas/code optimizations that shouldn't make the code less readable or improve overall readability. It also lists questions about unclear code segments.

Confusing naming of `safeValidateSupplyCap`

The private function `_approveRequest()` uses a new argument called `safeValidateSupplyCap` which represents whether the function should revert on overflowing max supply cap or return **`false`**. This naming is confusing especially following the `isSafe` argument that represents whether a check on price variation should be enforced.

I recommend naming the variable `revertAboveSupplyCap` instead and inverting its use in the code.

https://github.com/midas-apps/contracts/blob/e50803996f75b80dbdc4d57d7f3216c1f9570c81/contracts/DepositVault.sol#L552

Response:

The variable has been renamed in commit `0x6e98b97` but its use has not be inverted meaning the behaviour is now incorrect and the function does not revert when it should and reverts when it should not.

The use of the variable has been inverted (and fixed) in commit `0xba85967`.