OpenZeppelin | security

Morpho Upgradeable Token Audit

Morpho

November 5, 2024

Table of Contents

Summary

Type	DeFi
Timeline	From 2024-10-23 To 2024-10-24
Languages	Solidity

Total Issues	11 (4 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	0 (0 resolved)
Low Severity Issues	4 (1 resolved)
Notes & Additional Information	7 (3 resolved)

Scope

We audited the morpho-org/morpho-token-upgradeable repository at commit `0604649`.

In scope were the following files:

```
src
├── interfaces
│   ├── IDelegation.sol
│   └── IOptimismMintableERC20.sol
├── DelegationToken.sol
├── MorphoTokenEthereum.sol
├── MorphoTokenOptimism.sol
└── Wrapper.sol
```

Update: *The most recent commit hash examined after the fixes was commit* `daa7ba2`.

System Overview

The Morpho token project introduces an upgradeable ERC-20 token that supports on-chain voting delegation. The token operates across different blockchain environments, including Ethereum and Optimism, with each network having its own token contract. These contracts are designed to facilitate governance participation by allowing users to delegate their voting power directly or through signatures. In addition, the system includes a wrapper contract that supports the migration of legacy Morpho tokens to the new version, ensuring backward compatibility and smooth transitions for existing users.

The Ethereum version of the Morpho token mints 1 billion tokens to a migration wrapper contract, enabling the seamless upgrade of legacy token holders to the new version. The Optimism version interacts with a bridge contract to manage cross-chain token minting and burning, ensuring that token supply is consistent across both Ethereum and Optimism networks.

Security Model and Trust Assumptions

The system assumes that privileged roles will not act maliciously and ensures the proper validation of addresses and other critical security parameters across all smart contract functions.

Privileged Roles

The Morpho token has two privileged roles that are expected to operate securely and in the interest of token holders:

1. **Owner**: The owner role can perform critical operations, such as minting new tokens and upgrading the contract. It is assumed that the owner is trusted to handle these powers without malicious intent.

2. **Bridge (Optimism)**: On the Optimism network, the bridge has exclusive authority to mint and burn tokens. The security of the bridge contract and the accuracy of its address are central to maintaining the integrity of the token supply between Ethereum and Optimism.

Low Severity

L-01 Incomplete Docstrings

Throughout the codebase, there are multiple events and functions with incomplete docstrings. In particular:

- The `DelegateeChanged`, `DelegatedVotingPowerChanged`, `Mint`, and `Burn` events in `DelegationToken.sol`
- The `delegatee`, `delegatedVotingPower`, `delegationNonce`, `delegate`, and `delegateWithSig` functions in `DelegationToken.sol`
- The `mint` and `burn` functions in `MorphoTokenEthereum.sol`
- The `mint`, `burn`, `depositFor`, `withdrawTo`, and `underlying` functions in `MorphoTokenOptimism.sol`

Consider thoroughly documenting all functions/events (and their parameters or return values) that are part of a contract's public API. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

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

> *As the missing docstrings can be considered trivial we chose to not add them.*

L-02 Missing Docstrings

The `IDelegation` and `IOptimismMintableERC20` interfaces lack docstrings for their functions.

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).

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

> *We intentionally don't have docstrings in interfaces, so this issue is acknowledged.*

L-03 Floating Pragma

Pragma directives should be fixed to clearly identify the Solidity version with which the contracts will be compiled. The `DelegationToken.sol` file has the `^0.8.27` floating pragma directive.

Consider using fixed pragma directives.

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

> *This was a deliberate choice, the discussion can be seen here.*

L-04 Redundant Event Emissions

The token contracts `MorphoTokenOptimism` and `MorphoTokenEthereum` include functionality to enable minting and burning of tokens. When tokens are minted or burned, an event is emitted signifying the amount and address involved.

However, the standard mint/burn functionality included within the `ERC20` contract already emits an event when tokens are minted or burned signifying the change of balance of the user. As a result, these event emissions are redundant.

Consider removing redundant event emissions, in the case of the `Mint` and `Burn` events of the `MorphoTokenOptimism` and `MorphoTokenEthereum` tokens.

Update: *Resolved in pull request #71.*

Notes & Additional Information

N-01 Missing Named Parameters in Mappings

Since Solidity 0.8.18, developers can utilize named parameters in mappings. This means mappings can take the form of `mapping(KeyType KeyName? => ValueType ValueName?)`. This updated syntax provides a more transparent representation of a mapping's purpose.

Within `DelegationToken.sol`, multiple instances of mappings without named parameters were identified:

- The `_delegatee` state variable
- The `_delegatedVotingPower` state variable
- The `_delegationNonce` state variable

Consider adding named parameters to mappings in order to improve the readability and maintainability of the codebase.

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

> *We initially named the the mappings parameters be finally chose to remove the names as they might be confusing. We concluded not having them was better.*

N-02 Improper Security Contact Tag Usage

While a security contact has been provided in several contracts within the codebase, the current implementation uses the `@custom:contact` tag instead of the recommended `@custom:security-contact` tag. In addition, the `IOptimismMintableERC20` interface is missing a security contact tag entirely.

Consider using the `@custom:security-contact` convention as it has been adopted by the OpenZeppelin Wizard and the ethereum-lists for all Solidity files in the codebase.

Update: *Resolved in pull request #69.*

N-03 Constant Not Using `UPPER_CASE` Format

In `DelegationToken.sol`, the `ERC20DelegatesStorageLocation` constant has not been declared using `UPPER_CASE` format.

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

Update: *Resolved in pull request #67.*

N-04 Indecisive Licenses

Throughout the codebase, each contract and interface includes an indecisive SPDX license.

Consider specifying only one license to prevent possible licensing ambiguity.

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

> *As they are only few differences between GPL-2.0 and GPL-3.0 we chose to acknowledge the issue.*

N-05 Duplicate Event Emission in `_delegate`

The `_delegate` function currently does not check if the `oldDelegatee` is different from the `newDelegatee` before emitting the `DelegateeChanged` event. As a result, the event may be emitted even when there is no actual change in delegation. This could cause confusion for users or third-party services that rely on accurate event data.

Consider adding a conditional check to verify if `oldDelegatee` is different from `newDelegatee` before emitting the `DelegateeChanged` event. This would ensure that the event is only emitted when an actual change in delegation occurs, reducing potential confusion of event logs.

Update: *Acknowledged, not resolved.*

N-06 Direct Call to `_transferOwnership` in `initialize` Function

The `MorphoTokenOptimism` and `MorphoTokenEthereum` contracts directly call the `_transferOwnership` function within the `initialize` function instead of using the standard `__Ownable_init` method to initialize the `OwnableUpgradeable` contract. While this does not present a direct risk to the contract's functionality, following the standard initialization method (`__Ownable_init`) improves the readability of the code and ensures consistency across upgradable contracts using the OpenZeppelin libraries.

Consider replacing the direct call to `_transferOwnership` with a call to `__Ownable_init` in the `initialize` function. This change enhances the clarity of the contract and ensures that it follows the established initialization pattern for upgradable contracts.

Update: *Resolved in pull request #70.*

N-07 Gas Optimization in `_update` Function

The `_update` function in the `DelegationToken` contract calls the public `delegatee` function to retrieve the delegatee of the `from` and `to` accounts. This introduces unnecessary overhead as the `delegatee` function performs an additional lookup, whereas these values could be accessed directly from storage. By avoiding the extra function call and accessing the storage directly, the number of opcodes executed can be reduced, thereby lowering the gas costs associated with this operation.

Instead of calling the `delegatee` function, consider accessing the delegatee information for the `from` and `to` accounts directly from storage within the `_update` function. This optimization will reduce the gas usage of the function by minimizing the number of opcodes executed, leading to more efficient and cost-effective operations.

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

> *This optimisation is very small and adds lines of code, therefore we chose not to apply it.*

Conclusion

The Morpho upgradeable token introduces several features as an extension of the legacy Morpho token. It supports on-chain voting delegation and includes compatibility across both the Optimism and Mainnet Ethereum networks. It also includes a wrapper contract to allow for migration of the old tokens to the new one. The wrapper contract holds 1 billion Morpho tokens in order to facilitate minting and redemption via the legacy token.

This system was found to be highly secure as it made use of battle-tested functionality across existing codebases. No significant issues were found, and the only reported issues were of low and note-level severity. The Morpho Labs team was responsive in answering any protocol-related questions and provided thorough and prompt additional context where necessary.