OpenZeppelin | security

Fireblocks Upgradeable Tokens Audit - ERC20F

December 13, 2023

Table of Contents

Summary

Type	DeFi
Timeline	From 2023-11-13 To 2023-11-17
Languages	Solidity

Total Issues	11 (7 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	0 (0 resolved)
Low Severity Issues	1 (1 partially resolved)
Notes & Additional Information	10 (7 resolved)

Scope

We audited the Blockfold repository at commit e7003dfb.

Following the conclusion of the audit, Fireblocks has updated the license to `AGPL-3.0-or-later` in pull request #1 at commit 6d8327b. This update has had no impact on the security of the source code logic.

In scope were the following contracts:

```
contracts
├── ERC20F.sol
└── library
    ├── AccessRegistry
    │   ├── AccessListUpgradeable.sol
    │   ├── AccessRegistrySubscriptionUpgradeable.sol
    │   ├── AllowList.sol
    │   ├── DenyList.sol
    │   └── interface
    │       └── IAccessRegistry.sol
    ├── Errors
    │   ├── LibErrors.sol
    │   └── interface
    │       └── IERC20Errors.sol
    └── Utils
        ├── ContractUriUpgradeable.sol
        ├── MarketplaceOperatorUpgradeable.sol
        ├── PauseUpgradeable.sol
        ├── RoleAccessUpgradeable.sol
        ├── SalvageUpgradeable.sol
        └── TokenUriUpgradeable.sol
```

System Overview

The Fireblocks Upgradeable Tokens project implements upgradeable token contract, namely, `ERC20F`. This is compliant with the ERC-20 token standard, and additional functionality has been integrated for managing access control. The token facilitate the configuration of an access list for token transfers, providing the flexibility of implementing either an allowlist, where only addresses on the list can utilize the tokens, or a denylist, which restricts addresses on the list from interacting with the contract.

Privileged Roles

The protocol implements multiple privileged roles that are tied to the `ERC20F` contract.

The holder of `DEFAULT_ADMIN_ROLE` can:

- Grant roles to other addresses.

The holder of `UPGRADER_ROLE` can:

- Upgrade the implementation contracts to a new version.

The holder of `PAUSER_ROLE` can:

- Pause and unpause a contract.

The holder of `CONTRACT_ADMIN_ROLE` can:

- Change the used access list address.
- Update the contract URI.

The holder of `MINTER_ROLE` can:

- Mint tokens.

The holder of `BURNER_ROLE` can:

- Burn tokens.

The holder of `RECOVERY_ROLE` can:

- Recover tokens from an account that is missing access.

The holder of `SALVAGE_ROLE` can:

- Recover ETH and ERC-20 tokens that were sent to a token contract itself.

Trust Assumptions

- The holders of the `DEFAULT_ROLE_ADMIN` role are expected to be non-malicious and to act in the project's best interest.

Low Severity

L-01 Retrieving the Whole Access List Might Revert

The `AccessListUpgradeable` contract includes a `getAccessList` function which retrieves all values from the access list. This function utilizes the `values` function of `EnumerableSetUpgradeable.AddressSet`, copying the entire storage into memory. This might lead to problems when the access list is excessively large as the function execution remains susceptible to the block gas limit, resulting in the call getting reverted.

Consider implementing an additional function that can be used when the value set gets too large. This function should return only a subset or slice of the values from the entire set.

Update: *Acknowledged, partially resolved in merge request #15 at commit 130dbb5. The Fireblocks team stated:*

> *To address this issue, we've tackled it by incorporating additional documentation into the function. This decision was prompted by the fact that the purpose of getAccessList() is to serve as a support function tailored for fetching compact access lists in smaller projects. In the case of larger projects equipped with more extensive access lists, it is advisable to devise custom indexing logic to efficiently retrieve all addresses from the access list.*

Notes & Additional Information

N-01 Lack of Security Contact

Providing a specific security contact (e.g., an email address or ENS name) within a smart contract greatly simplifies the process for individuals to communicate if they identify a vulnerability in the code. This practice is beneficial as it permits the code owners to dictate the communication channel for vulnerability disclosure, eliminating the risk of miscommunication

or failure to report due to a lack of knowledge on how to do so. Additionally, if the contract incorporates third-party libraries and a bug surfaces in them, it becomes easier for the maintainers of those libraries to contact the appropriate person about the problem and provide mitigation instructions.

Consider adding a NatSpec comment containing a security contact on top of the contract definitions. Using the `@custom:security-contact` convention is recommended as it has been adopted by the OpenZeppelin Wizard and the ethereum-lists.

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

> *Because these contracts are crafted as templates for other companies and entities to deploy on a larger scale. There cannot be a single security contact. Therefore, when a company deploys one of these contracts, they are are welcome to provide their own security contact independently through the contract URI.*

N-02 Unnecessary Variable Cast

In the `AccessRegistrySubscriptionUpgradeable` contract, the `_accessRegistry` variable in the `_accessRegistryUpdate` function is unnecessarily casted.

To improve the overall clarity, intent, and readability of the codebase, consider removing any unnecessary casts.

Update: *Acknowledged, resolved in merge request #9 at commit 69c5add.*

N-03 Unused Imports

Throughout the codebase, there are multiple imports that are unused and could be removed:

- `import {LibErrors} from "../Errors/LibErrors.sol";` imports unused alias `LibErrors` in `ContractUriUpgradeable.sol`
- `import {LibErrors} from "../Errors/LibErrors.sol";` imports unused alias `LibErrors` in `MarketplaceOperatorUpgradeable.sol`
- `import {LibErrors} from "../Errors/LibErrors.sol";` imports unused alias `LibErrors` in `PauseUpgradeable.sol`
- `import {LibErrors} from "../Errors/LibErrors.sol";` imports unused alias `LibErrors` in `TokenUriUpgradeable.sol`

Consider removing unused imports to improve the overall clarity and readability of the codebase.

Update: *Acknowledged, resolved in merge request #6 at commit 4243399.*

N-04 Lack of `__gap` Variables

Throughout the codebase, there are multiple upgradeable contracts that do not have a `__gap` variable. For instance:

- The `AllowList` contract
- The `DenyList` contract
- The `ERC20F` contract

Note that the current implementation does not require the implementation of the `__gap` storage variable, as the listed contracts are not being inherited. However, it is still recommended to include it as it implies that the contract implementations might be upgraded and additional storage variables might be introduced in the future.

Consider adding a `__gap` storage variable to avoid future storage clashes in upgradeable contracts.

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

> *Contracts lacking `__gap` variables are not needed for our use case since upgrades are intended to occur through contract inheritance. In this process, the initial implementation, `V1`, is inherited by `V2`, where the new logic resides. This enables the addition of storage slots without any conflict with existing slots.*

N-05 Incorrect Storage Gap Size in `AccessListUpgradeable`

The `AccessListUpgradeable` contract uses a `__gap` storage variable of size `49`. It assumes that the `_accessList` storage variable occupies one storage slot, whereas in reality, it takes two storage slots. The `_accessList` is of type `EnumerableSetUpgradeable.AddressSet` which is a struct that uses 2 storage slots.

Consider changing the size of the `__gap` storage variable in the `AccessListUpgradeable` contract from `49` to `48`.

Update: *Acknowledged, resolved in merge request #5 at commit 519e415.*

N-06 Confusing Error Used

The `ERC20F` contract features a `recoverTokens` function designed to recover tokens from accounts lacking access. The problem arises when the account already has access to the tokens, resulting in transactions reverting with a potentially confusing error such as `ERC20InvalidReceiver`. In this context, since the account is not the receiver, the error should not imply receiver-related issues.

Consider adding a new error which indicates that the account has access to the tokens.

Update: *Acknowledged, resolved in merge request #10 at commit 6b3f311.*

N-07 Missing Input Validation

The burn function of `ERC20F` contract is missing a check to ensure that amount is not equal to 0.

Consider implementing the suggested validation in order to prevent unexpected behavior that may lead to potential attacks on the protocol.

Update: *Acknowledged, resolved in merge request #11 at commit 6953e70.*

N-08 Typographical Errors and Incorrect Comments

Consider addressing the following typographical errors and incorrect comments:

- The `AccessRegistrySubscriptionUpgradeable` contract implements the `__AccessRegsitrySubscription_init` function that has a typo in its name. It should be `__AccessRegistrySubscription_init`.
- The NatSpec comments that describe the `hasAccess` functions implemented in the `AccessListUpgradeable`, `AllowList`, and `DenyList` contracts are using the name of the parameter `claim` instead of `data`.

Update: *Acknowledged, resolved in merge request #4 at commit 7a5e68e.*

N-09 Gas Optimization

In the codebase, there are several instances of potential gas optimizations:

- The `_requireHasAccess` function, present in the `ERC20F` contract, redundantly checks the `isSender` value twice. It would be more efficient to initially verify if the account has access through the `accessRegistry` contract. If it does not have access, then based on the `isSender` value, either revert with an invalid sender or invalid receiver error.
- The codebase follows a common pattern of checking access after updating state variables. For instance, in the `RoleAccessUpgradeable` contract, the `revokeRole` function calls the `_authorizeRoleAccess` function after updating the role. To prioritize failing early and conserving gas, consider performing the access check before updating the state variables.

To improve gas consumption, readability and code quality, consider refactoring wherever possible.

Update: *Acknowledged, resolved in merge request #13 at commit f84408b.*

N-10 Missing Functionality For Rescuing ERC-721 and ERC-1155 Tokens

The `SalvageUpgradeable` abstract contract, inherited by the `ERC20F` contract, implements the logic for rescuing ETH and ERC-20 tokens. This is useful for when tokens are accidentally sent to the token contract itself instead of a valid recipient. However, the rescue logic only works for rescuing ETH and ERC-20 tokens, not ERC-721 or ERC-1155 tokens. As such, while the ERC-20 tokens accidentally sent to any of the three token contracts can be rescued, ERC-721 or ERC-1155 tokens cannot be.

Consider adding the functionality for rescuing ERC-721 and ERC-1155 tokens accidentally sent to the `ERC20F` contract.

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

> *Currently, there is no observed customer demand for salvaging an ERC-721 or ERC-1155 token. While this functionality could potentially be incorporated later through an upgrade, we are cautious about subjecting users to additional bytecode and gas fees for a feature that may not be utilized at the present moment.*

Conclusion

The Fireblocks Upgradeable Tokens project introduces upgradeable token contract adhering to ERC-20 standard, with added features for access control. This token allow the creation of an access list for transfers, enabling the implementation of either an allow list, restricting usage to specified addresses, or a deny list, preventing interactions with addresses on the list.

One low severity was issue was identified. In addition, various recommendations have been made to enhance the quality and documentation of the codebase. The Blockfold team was responsive and timely answered any questions we had about the codebase.