OpenZeppelin | security

Fireblocks Gasless Contracts Audit

Fireblocks

December 24, 2024

Table of Contents

Summary

Type	DeFi
Timeline	From 2024-12-02 To 2024-12-10
Languages	Solidity

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

Scope

We audited the fireblocks/fireblocks-smart-contracts repository at commit cf1bb85.

In scope were the following files:

```
contracts
├── gasless-contracts
│   ├── AccessRegistry
│   │   ├── AccessListUpgradeableGasless.sol
│   │   ├── AllowListGasless.sol
│   │   └── DenyListGasless.sol
│   ├── ERC1155FGasless.sol
│   ├── ERC20FGasless.sol
│   ├── ERC721FGasless.sol
│   ├── GaslessFactory.sol
│   └── TrustedForwarder.sol
├── gasless-upgrades
│   ├── AccessRegistry
│   │   ├── AllowListV2.sol
│   │   └── DenyListV2.sol
│   ├── ERC1155FV2.sol
│   ├── ERC20FV2.sol
│   └── ERC721FV2.sol
└── library
    ├── MetaTx
    │   └── ERC2771ContextInitializableUpgradeable.sol
    └── Proxy
        └── Proxy.sol
```

System Overview

The Fireblocks Gasless contracts are an extension of the Fireblocks Upgradeable Tokens, allowing them to support meta transactions by leveraging the ERC-2771 standard. These contracts inherit from the `ERC2771ContextInitializableUpgradeable` contract which allows users to optionally set a trusted forwarder, thereby providing users the flexibility to choose between gasless or non-gasless versions.

Along with the gasless versions of the utility tokens, access control, and access list, a `GaslessFactory` contract has also been introduced. This is a singleton contract that allows its users to deploy deterministic and non-deterministic contracts through meta transactions. In addition, a `TrustedForwarder` contract has been added, which directly inherits the OpenZeppelin contract library's `ERC2771Forwarder` contract. Both the `GaslessFactory` and `TrustedForwarder` contracts are non-upgradeable.

The repository also introduces another set of contracts called `Gasless upgrades`, which allow for upgrading the initial version of the contracts to the gasless version, and thereby introduce support for meta transactions. These contracts enforce that only the contracts on version 1 can upgrade to version 2, ensuring compatibility between the two versions and avoiding any storage collisions. A `Proxy` contract has also been introduced, which inherits the OpenZeppelin contract library's `ERC1967Proxy` contract, ensuring that Fireblocks' library users have a standardized version of the proxy contracts.

Security Model and Trust Assumptions

In addition to the roles inherited via the chain of inheritance, a `CONTRACT_ADMIN_ROLE` role has been defined in the following contracts. This role can update the trusted forwarder for each respective contract.

- `AccessListUpgradeableGasless` contract
- `AllowListV2` contract
- `DenyListV2` contract

- `ERC1155FGasless` contract
- `ERC1155FV2` contract
- `ERC20FGasless` contract
- `ERC20FV2` contract
- `ERC721FGasless` contract
- `ERC721FV2` contract

It is assumed that the accounts in charge of the above roles and actions always act in the intended manner.

Low Severity

L-01 Limitation in Asset Transfer Capabilities During Contract Deployment in `GaslessFactory`

The `GaslessFactory` contract facilitates the deployment of new contracts through two primary functions: `deploy` and `deployDeterministic`. The `deploy` function enables standard contract creation, while `deployDeterministic` allows for deterministic contract deployment using the `CREATE2` opcode. Despite their utility in contract deployment, both functions exhibit a limitation as they are not marked as `payable`. This restriction prevents users from sending native assets alongside the contract deployment transaction. Consequently, any scenario requiring the newly deployed contract to hold a native asset balance upon creation cannot be accommodated.

Furthermore, this limitation extends to the `postConfig` function, which is designed for calling functions of the newly deployed contracts as part of their initial configuration. Since `postConfig` also does not support sending native assets along with the function calls, it restricts the initialization capabilities, particularly for contracts whose setup functions require a native asset transfer.

This constraint not only limits the versatility of the `GaslessFactory` contract in deploying and configuring a wide range of contracts, but also complicates the deployment process for contracts designed to hold or manage native assets immediately upon creation. To overcome this limitation and enhance the `GaslessFactory` contract's functionality, consider making `deploy` and `deployDeterministic` functions `payable` and pass along the amount to the `create` or `create2` function. This will enable the two functions to accept native asset transfers as part of the contract deployment process and allow users to deploy contracts that require an initial native asset balance.

By implementing the aforementioned recommendations, the `GaslessFactory` contract will improve its utility and flexibility, accommodating a broader range of deployment and configuration scenarios, including those requiring immediate asset management capabilities.

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

> *This design was intentional as it keeps the `GaslessFactory` code clean and simple. We have no use cases for `payable` functions and want to avoid having extra code to handle edge cases like potential loss of funds or their recovery.*

L-02 Premature Event Emission in GaslessFactory's `_execute` Function

The `_execute` function of the `GaslessFactory` contract is designed to facilitate gasless transactions by delegating function calls to other contracts. However, an issue arises with the order of operations within this function, specifically concerning the emission of the `FunctionExecuted` event. The event is emitted prior to the actual execution of the delegated function call. This premature event emission leads to a scenario where the event logs a result of zero, regardless of the actual outcome of the function call that follows.

This behavior can mislead off-chain services or users monitoring these events, as the `FunctionExecuted` event suggests the completion of a function call without accurately reflecting its result. In a blockchain environment, where transparency and accuracy of operations are paramount, such discrepancies can undermine trust in the system's reliability.

Consider adjusting the sequence of operations within the `_execute` function. Specifically, the `FunctionExecuted` event should be emitted only after the delegated function call has been successfully executed. This ensures that the event accurately reflects the outcome of the function call, aligning with the expectations set by its emission. By implementing this change, the `GaslessFactory` contract will enhance its operational transparency and reliability, providing accurate feedback to the system's participants through event emissions.

Update: *Resolved in pull request #1 at commit 43a713b.*

Notes & Additional Information

N-01 Ambiguous Calls to Parent Contract

Throughout the codebase, multiple instances of ambiguous calls to parent contracts were identified:

- The `super._msgSender` in the `AccessListUpgradeableGasless` contract
- The `super._msgData` in the `AccessListUpgradeableGasless` contract
- The `super._msgSender` in the `AllowListV2` contract
- The `super._msgData` in the `AllowListV2` contract
- The `super._msgSender` in the `DenyListV2` contract
- The `super._msgData` in the `DenyListV2` contract
- The `super._msgSender` in the `ERC1155FV2` contract
- The `super._msgData` in the `ERC1155FV2` contract
- The `super._msgSender` in the `ERC20FGasless` contract
- The `super._msgData` in the `ERC20FGasless` contract
- The `super._msgSender` in the `ERC20FV2` contract
- The `super._msgData` in the `ERC20FV2` contract
- The `super._msgSender` in the `ERC721FV2` contract
- The `super._msgData` in the `ERC721FV2` contract
- The `super._msgSender` in the `GaslessFactory` contract
- The `super._msgData` in the `GaslessFactory` contract
- The `super._contextSuffixLength` in the `GaslessFactory` contract

To avoid ambiguous calls to parent contracts, consider explicitly specifying the parent contract's function being called.

Update: *Resolved in pull request #2 at commit 2ef4c95.*

N-02 Compatibility Issues of GaslessFactory Contract with ZkSync Era Deployment Mechanism

The `GaslessFactory` contract, designed for facilitating contract deployments and operations without gas fees, encounters significant functional discrepancies when deployed on the ZkSync Era platform. Two core issues underpin these discrepancies: the method of address derivation and the prerequisites for contract deployment concerning bytecode knowledge.

1. On Ethereum, the address of a new contract is derived through a deterministic process involving the deployer's address and the nonce (the number of transactions sent by the deployer's address). However, ZkSync Era employs a different mechanism for address derivation which does not align with Ethereum's method. This divergence means that the `GaslessFactory` contract's expectations for address derivation and, consequently, its ability to predict or interact with newly deployed contract addresses, are misaligned with the operational realities of ZkSync Era.

2. The deployment of contracts on ZkSync Era is uniquely characterized by its requirement for the hash of the contract's bytecode and the necessity for the compiler to be aware of this bytecode in advance. This requirement stands in contrast to Ethereum's deployment mechanism, where the bytecode is generated and provided at the time of deployment without prior knowledge needed by the compiler. The `GaslessFactory` contract's `deploy` function, which is designed to deploy contracts dynamically without precompiled bytecode knowledge, is inherently incompatible with ZkSync Era's deployment prerequisites. This incompatibility renders the `deploy` function ineffective on ZkSync Era, as it cannot fulfill the platform's requirement for bytecode awareness prior to deployment.

To address these compatibility issues and enable the `GaslessFactory` contract to function as intended on ZkSync Era, consider implementing the following modifications:

1. **Address Derivation Adaptation**: Implement a mechanism within the `GaslessFactory` contract to accommodate ZkSync Era's address derivation method. This may involve integrating ZkSync Era's address derivation logic or a contract call that can predict or determine addresses correctly within the ZkSync Era environment.

2. **Precompiled Bytecode Management**: Revise the contract deployment strategy to align with ZkSync Era's requirement for precompiled bytecode knowledge. This could involve strategies such as:
    - Maintaining a registry within the `GaslessFactory` contract of precompiled bytecodes for contracts intended for deployment.
    - Developing a tool or process for precompiling contract bytecodes and registering them with the `GaslessFactory` contract or another on-chain registry before any deployment attempts.
3. **Documentation and Developer Guidance**: Update the `GaslessFactory` contract's documentation to clearly describe its compatibility with ZkSync Era, including any limitations, prerequisites for deployment, and guidance on address derivation and precompiled bytecode management.

By implementing the aforementioned recommendations, the `GaslessFactory` contract can be adapted to operate effectively within the ZkSync Era ecosystem, thereby extending its utility to this emerging platform and ensuring its functionality aligns with the unique requirements of ZkSync Era's contract deployment and address management mechanisms.

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

> *We do not intend to make custom adjustments to our code for a specific blockchain. Additionally, we have no plans to deploy this factory on ZKSync. It is worth noting that deployment on ZKSync cannot happen accidentally, as it requires a separate and intentional compilation using the ZKSync compiler.*

N-03 Prefix Increment Operator `++i` Can Save Gas in Loops

Within `GaslessFactory.sol`, multiple opportunities for saving gas by using the prefix increment operator (`++i`) were identified:

- In line 135.
- In line 174.

Consider using the prefix increment operator (`++i`) instead of the post-increment operator (`i++`) in order to save gas. This optimization skips storing the value before the increment, as the return value of the expression is ignored.

Update: *Resolved in pull request #3 at commit a8942e9.*

N-04 Missing Docstrings

Within `Proxy.sol`, the docstring for the `Proxy` contract itself is missing.

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: *Resolved in pull request #4 at commit 5a1aa15.*

N-05 Lack of Security Contact

Providing a specific security contact (such as an email or ENS name) within a smart contract significantly simplifies the process for individuals to communicate if they identify a vulnerability in the code. This practice is quite 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. In addition, if the contract incorporates third-party libraries and a bug surfaces in those, it becomes easier for their maintainers to contact the appropriate person about the problem and provide mitigation instructions.

Given that these contracts are templates to be utilized by other entities, consider adding a security contact within the Fireblocks repository. An example of this can be found in the `SECURITY.md` of the OpenZeppelin Contracts library.

Update: *Resolved in pull request #8 at commit d3b7de0.*

N-06 Incremental Update Is Not Wrapped in an `unchecked` Block

Since Solidity version `0.8.0`, arithmetic operations include automatic checks for overflows and underflows, which increase gas costs. In scenarios where it is highly unlikely for a positively incrementing variable to overflow within a loop, using an `unchecked` block can optimize gas usage without compromising security. Before Solidity version `0.8.22`, developers manually implemented this optimization to bypass the additional overhead from automatic overflow checks.

Within `GaslessFactory.sol`, multiple opportunities for saving gas by using the `unchecked` block were identified:

- The `i++` increment operation in the `deploy` function
- The `i++` increment operation in the `deployDeterministic` function

Consider either updating the pragma version to `0.8.22` to leverage automatic overflow check optimizations or wrapping incremental updates in an `unchecked` block to save gas.

Update: *Resolved in pull request #5 at commit 1e0c6f5.*

N-07 Different Versions of OpenZeppelin Contracts Library Used

The `Proxy` contract uses version `4.9.3` of the OpenZeppelin Contracts library, whereas the `GaslessFactory` and `TrustedForwarder` contracts use version `5.0.2`.

To be consistent with the rest of the codebase, consider updating the dependency in the `Proxy` contract to version `5.0.2`.

Update: *Resolved in pull request #6 at commit 0e3be3a.*

N-08 Deployment Failure Revert Reason Not Propagated in `deployDeterministic`

The `GaslessFactory` contract utilizes the `deployDeterministic` function to create new contracts deterministically. This function is crucial for deploying contracts without requiring gas from the user, leveraging the `CREATE2` opcode for deterministic address generation. However, a limitation arises due to the use of version `5.0.x` of the OpenZeppelin Contracts library. In this version, the functionality to bubble up revert reasons during contract creation failures is absent. This feature was only introduced in version `5.1` of the OpenZeppelin Contracts library.

The absence of revert reason propagation means that when a contract deployment fails, the `deployDeterministic` function does not provide the caller with the specific reason for the failure. This lack of transparency can hinder debugging efforts and obscure potential vulnerabilities or logic errors in the contract code intended for deployment. It also impacts the user experience, as developers or users interacting with the `GaslessFactory` cannot easily ascertain the cause of deployment failures.

Consider upgrading the OpenZeppelin contracts library used by the `GaslessFactory` to version `5.1` or later. This newer version includes enhancements that allow revert reasons to be propagated back to the caller when a contract creation fails. Implementing this upgrade will make debugging more straightforward and improve the overall security posture by ensuring that the causes of deployment failures are transparent and actionable.

Update: *Resolved in pull request #7 at commit ddd1e4a.*

Conclusion

The Fireblocks Gasless contracts demonstrate a well-structured implementation of meta-transaction capabilities, integrating the ERC-2771 standard seamlessly within the `ERC-20`, `ERC-721`, and `ERC-1155` Fireblocks Upgradeable Tokens. The codebase showcases a comprehensive approach to enabling gasless transactions while maintaining compatibility and flexibility for users, supported by robust design principles such as deterministic and non-deterministic contract deployments through the `GaslessFactory` contract. The codebase was found to be well-written and thoroughly documented, while the Fireblocks team timely answered any questions we had about the codebase.