RENASCENCE

EVM Gateway Protocol Audit Report

Version 2.0

Audited by:

MiloTruck

alexxander

August 12, 2024

Contents

1 Introduction

1.1 About Renascence

Renascence Labs was established by a team of experts including HollaDieWaldfee, MiloTruck, alexxander and bytes032.

Our founders have a distinguished history of achieving top honors in competitive audit contests, enhancing the security of leading protocols such as Reserve Protocol, Arbitrum, MaiaDAO, Chainlink, Dodo, Lens Protocol, Wenwin, PartyDAO, Lukso, Perennial Finance, Mute and Taurus.

We strive to deliver tailored solutions by thoroughly understanding each client's unique challenges and requirements. Our approach goes beyond addressing immediate security concerns; we are dedicated to fostering the enduring success and growth of our partners.

More of our work can be found here.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an 'as-is' and 'as-available' basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	High	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

1.3.1 Impact

- High - Funds are **directly** at risk, or a **severe** disruption of the protocol's core functionality
- Medium - Funds are **indirectly** at risk, or **some** disruption of the protocol's functionality
- Low - Funds are **not** at risk

1.3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

2 Executive Summary

2.1 About EVM Gateway Protocol

The Gateway Protocol is a standard that allows smart contracts to add access control constraints, requiring that a user has a valid Gateway Token (GT) in order to interact with the smart contract.

Users can create a Gatekeeper Network to which Gatekeepers are added and can issue GT tokens for that network. A primary authority address is selected, which can configure the network's settings and add or remove Gatekeepers.

The Gatekeepers within a network can freeze, unfreeze, revoke, burn, and manage the expiration time of GTs. Each Gateway Network also includes a global namespace of token flags, which can be set by Gatekeepers when issuing a token. The flags are stored as a bitmask and can be used to indicate additional information about the token.

2.2 Overview

Project	EVM Gateway Protocol
Repository	gateway-protocol-evm
Commit Hash	7fcc3be770af...
Mitigation Hash	89d38262a9b0...
Date	17 July 2024 - 24 July 2024

2.3 Issues Found

Severity	**Count**
High Risk	2
Medium Risk	6
Low Risk	7
Informational	7
Total Issues	**22**

3 Findings Summary

ID	Description	Status
H-1	Not clearing the `_nextPrimaryAuthoritys` mapping allows a network takeover for a re-opened network	Resolved
H-2	Signature replay in `FlexibleNonceForwarder`	Resolved
M-1	Issued tokens will remain valid after closing a network	Acknowledged
M-2	Function `GatewayToken._existsAndActive()` can revert instead of returning `false`	Resolved
M-3	Missing validation that the new `expiration` of a token is greater than `block.timestamp`	Resolved
M-4	Function `GatewayToken.unfreeze()` should be `payable` to handle ETH fees	Resolved
M-5	Transfer of shares in `GatewayStaking` should be disabled	Resolved
M-6	Function `GatewayStaking.withdrawStake()` should prevent withdrawals by active gatekeepers that will reduce their shares below `GLOBAL_MIN_GATEKEEPER_STAKE`	Resolved
L-1	The gatekeeper's array in a network is unbounded	Resolved
L-2	Function `GatewayNetwork.addGatekeepers()` will always revert	Resolved
L-3	Front-running can prevent users to create a network with the name they want	Acknowledged
L-4	Missing input validation for `GatekeeperNetworkData.networkFee` in `GatewayNetwork.createNetwork()`	Resolved
L-5	`GatewayToken.setExpiration()` should revert if `passExpireDurationInSeconds` hasn't passed since minting the token	Resolved
L-6	Some `GatewayToken` functions are missing a `checkGatekeeperHasMinimumStake` modifier	Resolved
L-7	Missing return values in functions `GatewayStaking.depositStake()` and `GatewayStaking.withdrawStake()`	Resolved
I-1	The reverting `receive()` function in `GatewayNetwork` is redundant	Resolved
I-2	Redundant `address(0)` check in modifier `onlyPrimaryNetworkAuthority`	Acknowledged
I-3	Redundant `address(0)` check for the `primaryAuthority` in several functions	Resolved
I-4	Redundant setting of `DEFAULT_ADMIN_ROLE` to be the role admin for `NETWORK_FEE_PAYER_ROLE`	Acknowledged
I-5	Missing calls to `_disableInitializers()` in the constrcutors of several contracts	Resolved

ID	Description	Status
I-6	Code improvements	Acknowledged
I-7	`ForwardRequest` should contain a deadline parameter	Acknowledged

4 Findings

High Risk

[H-1] Not clearing the `_nextPrimaryAuthoritys` mapping allows a network takeover for a re-opened network

Context:

- GatewayNetwork.sol#L108-L115
- GatewayNetwork.sol#L183-L189

Description: The `_nextPrimaryAuthoritys` mapping isn't cleared when a network is closed. This makes it possible for someone to become the primary authority of a network after it is closed. For example:

- User calls `updatePrimaryAuthority()`, which sets `_nextPrimaryAuthoritys[networkName]` to himself.
- User calls `closeNetwork()` to delete the network.
- User calls `claimPrimaryAuthority()`, which sets himself as the primary authority.

This would prevent a network with the same name from being deployed as `primaryAuthority` is no longer the zero address. Additionally, the user can call functions in this contract even when the network is closed. In the case where a new network with the same name is created, using `createNetwork()`, the user can call `claimPrimaryAuthority()` to take over the new network.

Recommendation: When `closeNetwork()` is called, clear the `_nextPrimaryAuthoritys` mapping as well:

```
  delete _networks[networkName];
+ delete _nextPrimaryAuthoritys[networkName];
```

Identity: Fixed in commit 49d177b.

Renascence: Verified, the recommended fix was implemented.

[H-2] Signature replay in `FlexibleNonceForwarder`

Context:

- FlexibleNonceForwarder.sol#L85-L112

Description: The `FlexibleNonceForwarder` contract allows for signatures to be replayed, consider the following example:

- Assume `currentNonce = 1`.
- A signature with `nonce = 2` is executed:
  - `currentNonce == req.nonce` is `false`.
  - `sigs[signature]` is also `false` so this check passes.
  - `sigs[signature]` is set to `true` below to mark that the signature was seen.
- Another signature with `nonce = 1` is executed:

    - `currentNonce == req.nonce` is `true`, so `currentNonce` is incremented to `2`.
- The same signature with `nonce = 2` is executed again:
    - `currentNonce == req.nonce` is `true`, so `_verifyFlexibleNonce()` doesn't revert.

In the example above, the same signature with `nonce = 2` was executed twice.

Recommendation: It will be safer to use a nonce that is incremented after use.

Identity: Fixed in commit 570b906.

Renascence: Verified, flexible nonces have been removed in favor of nonces that are incremented sequentially.

Medium Risk

[M-1] Issued tokens will remain valid after closing a network

Context:

- GatewayToken.sol#L561
- GatewayToken.sol#L327

Description: The function `GatewayToken._existsAndActive()` doesn't check if a token's network still exists, so tokens remain valid even after a network has been closed using `closeNetwork()`. More specifically, calling `verifyToken()` will still return `true`.

Another issue is that tokens from a previous network that has been closed will "carry over" to a new network with the same name. For example:

- `GatewayNetwork.createNetwork()` is called with `network.name = "NETWORK"`.
- A gateway token is minted using `GatewayToken.mint()`. We call this token A.
- `GatewayNetwork.closeNetwork()` is called to close the network.
- After some time, `GatewayNetwork.createNetwork()` is called again with the same network name.
- Now, token A is still valid and is part of the new network.

Recommendation: Tokens should be invalidated upon `GatewayNetwork.closeNetwork()`, unless the intended design is that tokens remain valid after a network has been closed.

Identity: Acknowledged.

Renascence: This issue has been acknowledged. Do note that the risks of a new network being created with the same name still exists, so previous tokens will be carried over to the new network.

[M-2] Function `GatewayToken._existsAndActive()` can revert instead of returning `false`

Context:

- GatewayToken.sol#L570
- ERC3525Upgradeable.sol#L292-L294

Description: If `tokenId` does not exist, the call to `slotOf()` will revert as `slotOf()` calls `_requireMinted()`, which reverts if `tokenId` does not exist: ERC3525Upgradeable.sol#L292-L294

```
function _requireMinted(uint256 tokenId_) internal view virtual {
    require(_exists(tokenId_), "ERC3525: invalid token ID");
}
```

As such, if `tokenId` does not exist, `_existsAndActive()` will revert instead of returning `false`.

Since `verifyToken(uint)` calls `_existsAndActive()`, it will also revert if `tokenId` doesn't exist instead of returning `false`.

Recommendation: The reverting behavior of the function `GatewayToken._existsAndActive()` should be documented to prevent smart contract and off-chain integration errors with `GatewayToken`.

Identity: Fixed in commit 570b906.

Renascence: Verified, this behavior is now documented in a comment.

[M-3] Missing validation that the new `expiration` of a token is greater than `block.timestamp`

Context:

- GatewayToken.sol#L218
- GatewayToken.sol#L434-L439

Description: There is no validation that the `expiration` in `GatewayToken.mint()` is greater than `block.timestamp`. If the gatekeeper sets `expiration` to a timestamp smaller than the current time, the created gateway token will instantly expire. Note that `GatewayToken.setExpiration()` also doesn't check that `timestamp` is greater than `block.timestamp`.

Recommendation: Check that `_expirations[tokenId]` cannot be set to a past timestamp in both functions:

```
- } else if (expiration > 0) {
+ } else if (expiration > block.timestamp) {
      _expirations[tokenId] = expiration;
  }
```

In `setExpiration()`, add the following check:

```
require(timestamp == 0 || timestamp > block.timestamp, "expiry is in the past");
```

Identity: Fixed in commit 49d177b.

We decided to keep the logic the same in `GatewayToken.setExpiration()`. We want to support gatekeepers being able to set `expiration` in the past. This is how gatekeepers can forcefully expire passes if needed (eg. if they found fraud with someone's identity verification).

Renascence: Verified, `mint()` can only be called with `expiration` as either `0` or a timestamp in the future.

[M-4] Function `GatewayToken.unfreeze()` should be `payable` to handle ETH fees

Context:

- GatewayToken.sol#L255-L264

Description: The function `GatewayToken.unfreeze()` calls `_handleCharge()` to handle the gatekeeper's and the gateway network's fees, it should be marked as `payable` to be able to handle ETH fees.

Recommendation:

```
- function unfreeze(uint tokenId, ChargeParties memory partiesInCharge) external
virtual {
+ function unfreeze(uint tokenId, ChargeParties memory partiesInCharge) external
payable virtual {
```

Identity: Fixed in commit 570b906.

Renascence: Verified, the recommended fix was implemented.

[M-5] Transfer of shares in `GatewayStaking` should be disabled

Context:

- ERC20.sol#L104
- ERC20.sol#L149

Description: A gatekeeper who has staked funds in order to pass `GLOBAL_MIN_GATEKEEPER_STAKE` can transfer their shares to another address, which can then become a gatekeeper. The same operation can be performed to bypass the modifier `checkGatekeeperHasMinimumStake` by multiple gatekeepers while there is only a single amount of `GLOBAL_MIN_GATEKEEPER_STAKE` staked.

Recommendation: Modify the functions `transfer()` and `transferFrom()` so that they revert.

```
+    function transferFrom(address from, address to, uint256 value) public override
returns (bool) {
+        revert VaultMethodNotImplemented();
+    }
+
+    function transfer(address to, uint256 value) public override returns (bool) {
+        revert VaultMethodNotImplemented();
+    }
```

Identity: Fixed in commit 570b906.

Renascence: Verified, both `transfer()` and `transferFrom()` have been overriden to revert in `IGatewayStaking`.

[M-6] Function `GatewayStaking.withdrawStake()` should prevent withdrawals by active gatekeepers that will reduce their shares below `GLOBAL_MIN_GATEKEEPER_STAKE`

Context:

- GatewayStaking.sol#L29-L35

Description: There is no minimum lock duration enforced upon `GatewayStaking.withdrawStake()` in this contract, therefore gatekeepers are able to perform the following actions:

- Call `depositStake()` to deposit the minimum stake.
- Perform actions that use `hasMinimumGatekeeperStake()` and require a minimum stake.
- Immediately call `withdrawStake()` afterwards to unstake.

A malicious gatekeeper could even use a flashloan for this, so they don't actually need to own the amount of funds required.

Recommendation: As long as a gatekeeper is active, his shares shouldn't fall below `GLOBAL_MIN_GATEKEEPER_STAKE`.

Identity: Fixed in commit 570b906.

Renascence: This fix doesn't work as a staker can directly call ERC4626's `deposit()` or `redeem()` to avoid the `_lastDepositTimestamp` check, since they are both declared `public`.

For example, a staker could:

- Call `redeem()` instead of `withdrawStake()`, which doesn't enforce the 7-day timelock.
- Call `deposit()` with the `receiver` as another address, causing `_lastDepositTimestamp` to not be set for the `receiver` address.

I believe the most optimal implementation would be to revert the changes to `depositStake()` and `withdrawStake()`, and override `deposit()` and `redeem()` as such:

```
function deposit(uint256 assets, address) public override returns (uint256) {
    _lastDepositTimestamp[msg.sender] = block.timestamp;
    return super.deposit(assets, msg.sender);
}

function redeem(uint256 shares, address, address) public override returns (uint256) {
    // check if time lock has expired
    uint256 lastDepositTimestamp = _lastDepositTimestamp[msg.sender];
    if (block.timestamp < lastDepositTimestamp + DEPOSIT_TIMELOCK_TIME) {
        revert GatewayStaking_Withdrawal_Locked(...);
    }

    return super.redeem(shares, msg.sender, msg.sender);
}
```

Identity: Fixed in commit 89d3826.

Renascence: Verified, the fix recommended above was implemented.

Low Risk

[L-1] The gatekeepers array in a network is unbounded

Context:

- GatewayNetwork.sol#L140
- GatewayNetwork.sol#L236-L238
- GatewayNetwork.sol#L157-L164

Description: The number of gatekeepers that can be added in a network is unbounded. If too many gatekeepers are added, looping through all gatekeepers might consume too much gas, causing a transaction to revert due to exceeding the block gas limit.

The functions `GatewayNetwork.removeGatekeeper()` and `GatewayNetwork.isGatekeeper()` could be permanently DOSed. The functions from `GatewayToken` that verify if a gatekeeper is active would be DOSed as well.

Recommendation: In `addGatekeeper()`, add an upper limit on how many gatekeepers can be added:

```
  GatekeeperNetworkData storage networkData = _networks[networkName];
+ require(networkData.gatekeepers.length < 20, "Too many gatekeepers");
```

Identity: Fixed in commit 570b906.

Renascence: Verified, a maximum of 21 gatekeepers can be added now.

[L-2] Function `GatewayNetwork.addGatekeepers()` will always revert

Context:

- GatewayNetwork.sol#L118-L121

Description: In `GatewayNetwork.addGatekeepers()`, the call `this.addGatekeeper(gatekeepers[i], networkName);` will perform an external call to `GatewayNetwork.addGatekeeper()`. This causes the `msg.sender` to become the `GatewayNetwork` contract and therefore the modifier `onlyPrimaryNetworkAuthority` will revert.

Recommendation: Change `addGatekeeper` from `external` to `public` and call `addGatekeeper` directly:

```
- this.addGatekeeper(gatekeepers[i], networkName);
+ addGatekeeper(gatekeepers[i], networkName);
```

Identity: Fixed in commit 570b906.

Renascence: Verified, the recommended fix was implemented.

[L-3] Front-running can prevent users to create a network with the name they want

Context:

- GatewayNetwork.sol#L43-L60

Description: Since networks are uniquely identified by their `networkName` and multiple networks cannot have the same name, an attacker can force a call to `createNetwork()` to revert with front-running.

For example:

- User calls `createNetwork()` with a certain `network.name`.
- Attacker front-runs the user to call `createNetwork()` with the same `network.name`, but with himself as `network.primaryAuthority`.
- Attacker's call is executed first, creating the network with himself as the primary authority.
- User's call to `createNetwork()` reverts.

As seen from above, it is no longer possible for the user to create a network with the same name. This is similar to domain squatting - legitimate users might be blocked from creating networks with the name they want.

Recommendation: Consider collecting a fee for creating networks. This ensures that attackers are disincentivized from occupying network names and/or DOSing the `createNetwork()` function.

Identity: Acknowledged.

Renascence: This issue has been acknowledged.

[L-4] Missing input validation for `GatekeeperNetworkData.networkFee` in `GatewayNetwork.createNetwork()`

Context:

- GatewayNetwork.sol#L43-L60

Description: The input validation for `network.networkFee` in `GatewayNetwork.createNetwork()` is missing. In `GatewayNetwork.updateFees()`, the function checks that all fees are lower than `MAX_FEE_BPS` before setting the fees:

GatewayNetwork.sol#L201-L205

```
// checks
require(fees.issueFee <= MAX_FEE_BPS, "Issue fee must be below 100%");
require(fees.refreshFee <= MAX_FEE_BPS, "Refresh fee must be below 100%");
require(fees.expireFee <= MAX_FEE_BPS, "Expiration fee must be below 100%");
require(fees.freezeFee <= MAX_FEE_BPS, "Freeze fee must be below 100%");
```

Therefore, it is possible to call `createNetwork()` to create a network with fees greater than 100%. Depending on which fee is above 100%, this will DOS functions in the `GatewayToken` contract as `ChargeHandler.handleCharge()` will revert:

ChargeHandler.sol#L104-L105

```
uint256 networkFee = (charge.value * charge.networkFeeBps) / 10_000;
uint256 gatekeeperFee = charge.value - networkFee;
```

Recommendation: In `createNetwork()`, check if the fees in `network.networkFee` are not greater than 100% by adding the following checks:

```
require(network.networkFee.issueFee <= MAX_FEE_BPS, "Issue fee must be below 100%");
require(network.networkFee.refreshFee <= MAX_FEE_BPS, "Refresh fee must be below
100%");
require(network.networkFee.expireFee <= MAX_FEE_BPS, "Expiration fee must be below
100%");
require(network.networkFee.freezeFee <= MAX_FEE_BPS, "Freeze fee must be below 100%");
```

Identity: Fixed in commit 49d177b.

Renascence: Verified, the recommended fix was implemented.

[L-5] `GatewayToken.setExpiration()` should revert if `passExpireDurationInSeconds` hasnt passed since minting the token

Context:

- GatewayToken.sol#L271-L281
- GatewayToken.sol#L434-L439

Description: The `GatewayToken.setExpiration()` function doesn't check if `passExpireDurationInSeconds` is set in the network by the primary authority. As such, if the previous expiry timestamp was set to `block.timestamp + passExpireDurationInSeconds` in `mint()`, any gatekeeper can simply override the expiry duration specified by the primary authority. They could even cancel the expiry duration by calling `setExpiration()` with `timestamp = 0`.

Recommendation: Add a check that `block.timestamp + passExpireDurationInSeconds` has passed before gatekeepers can call `setExpiration()`.

Identity: Fixed in commit 570b906.

Renascence: The following check was added:

```
uint networkDefaultExpiration = IGatewayNetwork(_gatewayNetworkContract).getNetwork(n
etwork).passExpireDurationInSeconds;
uint tokenExpiration = _expirations[tokenId];

// If network set a tokens expiration, it can not be updated until the
networkExpiration expires
if (networkDefaultExpiration > 0) {
    require(block.timestamp >= tokenExpiration, "Network expiration must expire");
}
```

The issue with this fix is the `block.timestamp >= tokenExpiration` check will always be active as long as `passExpireDurationInSeconds` is non-zero, even if the network expiration period has passed.

For example:

- Assume `passExpireDurationInSeconds` is 7 days.
- Gatekeeper mints a token at T days.
- After T+7 days, gatekeeper calls `setExpiration()` to set the expiration timestamp to T+14 days:
- At T+10 days, the gatekeeper tries to call `setExpiration()` to adjust the expiration timestamp:
  - `networkDefaultExpiration` is 7 days, which passes the `networkDefaultExpiration > 0` check.
  - `tokenExpiration` is T+14 days, so the `block.timestamp >= tokenExpiration` check fails and the function reverts.

Essentially, whenever the network's `passExpireDurationInSeconds` is set, gatekeepers can only call `setExpiration()` once the expiration timestamp has passed.

Identity: Acknowledged, if a network sets the original expiration, then gatekeepers need to wait for that to expire before they can change the expiration.

[L-6] Some `GatewayToken` functions are missing a `checkGatekeeperHasMinimumStake` modifier

Context:

- GatewayToken.sol#L232-L238
- GatewayToken.sol#L245-L249
- GatewayToken.sol#L288-L291
- GatewayToken.sol#L189-L192

Description: Currently, `_handleCharge()` has the `checkGatekeeperHasMinimumStake` modifier, therefore, functions `mint()`, `unfreeze()` and `setExpiration()` require that a gatekeeper has the minimum stake where the minimum stake could have increased after adding the gatekeeper. However, functions `revoke()`, `freeze()`, `setBitmask()` and `burn()` do not require that the gatekeeper has the minimum stake.

Recommendation: Unless intended by design, add the modifier `checkGatekeeperHasMinimumStake` to functions `revoke()`, `freeze()`, `setBitmask()` and `burn()`.

Identity: Fixed in commit 570b906. Note that `setBitmask()` does not need to check the gatekeeper's minimum stake.

Renascence: Verified, the recommended fix was implemented.

[L-7] Missing return values in functions `GatewayStaking.depositStake()` and `GatewayStaking.withdrawStake()`

Context:

- GatewayStaking.sol#L23-L27
- GatewayStaking.sol#L29-L35

Description: The function `GatewayStaking.depositStake()` should return the amount of shares received upon deposit and the function `GatewayStaking.withdrawStake()` should return the amount of assets that were withdrawn. Currently, the return values of those functions will always be 0.

Recommendation:

```
  function depositStake(uint256 assests) public override returns(uint256) {
     // Deposit stake using ERC-4626 deposit method
     require(assests > 0, "Must deposit assets to receive shares");
-    deposit(assests, msg.sender);
+    return deposit(assests, msg.sender);
  }

  function withdrawStake(uint256 shares) public override returns (uint256) {
     // checks
     require(shares > 0, "Must burn shares to receive assets");

     // Redeem stake using ERC-4626 redeem method
-    redeem(shares, msg.sender, msg.sender);
+    return redeem(shares, msg.sender, msg.sender);
  }
```

Identity: Fixed in commit 570b906.

Renascence: Verified, the recommended fix was implemented.

Informational

[I-1] The reverting `receive()` function in `GatewayNetwork` is redundant

Context:

- GatewayNetwork.sol#L272-L274

Description: Having a receive function that reverts is redundant. If this contract didn't have a receive function in the first place, attempting to transfer ETH to this contract normally would also revert.

Recommendation:

```
-    /**
-     * @dev Fallback function to receive ETH disabled
-     */
-    receive() external payable {
-        revert GatewayNetwork_Cannot_Be_Sent_Eth_Directly();
-    }
-
```

Identity: Fixed in commit 570b906.

Renascence: Verified, the recommended fix was implemented.

[I-2] Redundant `address(0)` check in modifier `onlyPrimaryNetworkAuthority`

Context:

- GatewayNetwork.sol#L27

Description: In the modifier `GatewayNetwork.onlyPrimaryNetworkAuthority()`, the `_networks[networkName].primaryAuthority != address(0)` check is redundant - if `primaryAuthority` is `address(0)`, it is not possible for the second check to pass and the modifier will revert.

Recommendation: Consider removing this check:

```
- require(_networks[networkName].primaryAuthority != address(0), "Network does not
exist");
  require(msg.sender == _networks[networkName].primaryAuthority, "Only the primary
  authority can perform this action");
```

Identity: Acknowledged.

Renascence: This issue has been acknowledged.

[I-3] Redundant `address(0)` check for the `primaryAuthority` in several functions

Context:

- GatewayNetwork.sol#L109
- GatewayNetwork.sol#L125
- GatewayNetwork.sol#L147
- GatewayNetwork.sol#L192

Description: Checking that `primaryAuthority != address(0)` in a function with the `onlyPrimaryNetworkAuthority` modifier is redundant as `primaryAuthority` must be the caller.

Recommendation: Consider removing the `_networks[networkName].primaryAuthority != address(0)` check in the following functions:

- `closeNetwork()`
- `addGatekeeper()`
- `removeGatekeeper()`
- `updatePassExpirationTime()`

Identity: Fixed in commit 570b906.

Renascence: Verified, the recommended fix was implemented.

[I-4] Redundant setting of `DEFAULT_ADMIN_ROLE` to be the role admin for `NETWORK_FEE_PAYER_ROLE`

Context:

- GatewayNetwork.sol#L37

Description: Setting the admin of `NETWORK_FEE_PAYER_ROLE` to `DEFAULT_ADMIN_ROLE` is redundant as the role admin is already set to `DEFAULT_ADMIN_ROLE` by default.

Recommendation: Consider removing the call to `_setRoleAdmin()`:

```diff
  // Allow contract deployer to set NETWORK_FEE_PAYER_ROLE role
  _grantRole(DEFAULT_ADMIN_ROLE, 0, owner);
- _setRoleAdmin(NETWORK_FEE_PAYER_ROLE, 0, DEFAULT_ADMIN_ROLE);
```

Identity: Acknowledged.

Renascence: This issue has been acknowledged.

[I-5] Missing calls to `_disableInitializers()` in the constrcutors of several contracts

Context:

- GatewayNetwork.sol#L13
- Gatekeeper.sol#L10

Description: The best practice in contracts that inherit from `Initializable` is to disable the initializers since if left uninitialized they can be invoked in the implementation contract by an attacker. For example, there is a past vulnerability disclosure that demonstrates how initializers getting called in the implementation can lead to contract takeover where the attacker can appoint an owner and would self-destruct the implementation, therefore, bricking the Proxy: OZ post-mortem. Although this issue has been fixed from OZ version 4.3.2 it's still best practice to call `Initializable._disableInitializers()` in a constructor in the implementation.

```
# Initializable.sol

* [CAUTION]
 * ====
 * Avoid leaving a contract uninitialized.
 *
 * An uninitialized contract can be taken over by an attacker. This applies to both a
 proxy and its implementation
 * contract, which may impact the proxy. To prevent the implementation contract from
 being used, you should invoke
 * the {_disableInitializers} function in the constructor to automatically lock it
 when it is deployed:
 *
```

Recommendation: Add a constructor with a call to `_disableInitializers()` in contracts `GatewayNetwork` and `Gatekeeper`.

```
+    constructor() {
+        _disableInitializers();
+    }
```

Identity: Fixed in commit 49d177b.

Renascence: Verified, the recommendation was implemented.

[I-6] Code improvements

Context:

1. GatewayToken.sol#L591-L593
2. GatewayToken.sol#L484-L494
3. GatewayToken.sol#L151-L153
4. GatewayToken.sol#L274
5. GatewayStaking.sol#L42 / IGatewayStaking.sol#L50
6. GatewayStaking.sol#L15
7. Gatekeeper.sol#L38
8. BitMask.sol#L28
9. ChargeHandler.sol#L65-L67
10. FlexibleNonceForwarder.sol#L67
11. GatewayNetwork.sol#L154-L164
12. FlexibleNonceForwarder.sol#L51

Description / Recommendation:

1. The function `GatewayToken._checkSenderRole()` isn't used anywhere in the contract and can be removed.

```
-    /// @dev Checks if the sender has the specified role on the specified network and revert otherwise
-    function _checkSenderRole(bytes32 role, uint network) internal view {
-        _checkRole(role, network, _msgSender());
-    }
-
```

2. Using a `try{} catch{}` in `GatewayToken._handleCharge()` is redundant because the `catch{}` block will also revert. Consider performing the call normally as `_handleCharge()` is expected to revert if handling fees fails for some reason:

```
- try _chargeHandler.handleCharge{value: msg.value}(charge, networkId) {
-     // done
- } catch (bytes memory reason) {
-     // Rethrow the custom error from the charge handler
-     // Using inline assembly here avoids the need to parse the revert reason
-     // solhint-disable-next-line no-inline-assembly
-     assembly {
-         revert(add(32, reason), mload(reason))
-     }
- }
+ _chargeHandler.handleCharge{value: msg.value}(charge, networkId);
```

3. The constructor of `GatewayToken` reverts if a trusted forwarder is `address(0)`. For code consistency, `GatewayToken.addForwarder()` could also revert with `Common__MissingAccount()` if the supplied `forwarder` is `address(0)`.

```
+        if (forwarder == address(0)) {
+            revert Common__MissingAccount();
+        }
```

4. In `GatewayToken.setExpiration()`, `network` could be passed to `GatewayToken._checkGatekeeper()` instead of calling `slotOf(tokenId)` again.

```
  uint network = slotOf(tokenId);
- _checkGatekeeper(slotOf(tokenId));
+ _checkGatekeeper(network);
```

5. The function `balanceOf()` is public, therefore it can be called directly, without casting to `address(this)`.

```
- return ERC20(address(this)).balanceOf(staker) >= GLOBAL_MIN_GATEKEEPER_STAKE;
+ return balanceOf(staker) >= GLOBAL_MIN_GATEKEEPER_STAKE;
```

```
- require(ERC20(address(this)).balanceOf(msg.sender) >= shares, "...");
+ require(balanceOf(msg.sender) >= shares, "...");
```

6. The upgradeable versions of `ERC4626` and `ERC20` aren't used in `GatewayStaking`. `ERC4626` will still work as expected since its state variables are immutable, but the `name` and `symbol` in `ERC20` will be empty. Consider using `ERC4626Upgradeable` instead.

7. In `Gatekeeper.initializeGatekeeperNetworkData()`, `lastFeeUpdateTimestamp` will be `0` in storage by default. Consider removing the assignment operation.

```
- _gatekeeperStates[gatekeeper][networkName].lastFeeUpdateTimestamp = 0;
```

8. The `BitMask.checkBit()` function can refactored to use the `_ONE` constant:

```
- return (self & (uint256(1) « index)) > 0;
+ return (self & (_ONE « index)) > 0;
```

9. The `ChargeHandler.setRole()` function is redundant, the owner that has `DEFAULT_ADMIN_ROLE` can simply call `grantRole()` to give addresses the `CHARGE_CALLER_ROLE`. Consider removing this function:

```
- function setRole(bytes32 role, address recipient) external
onlyRole(DEFAULT_ADMIN_ROLE) {
-     _setupRole(role, recipient);
- }
```

10. In `FlexibleNonceForwarder.execute()`, use `req.gas / 63` instead of `req.gas / 64`, see the following explanation from OpenZeppelin's implementation or the linked article.

11. You could use the cached `currentGatekeepers` storage variable instead of `_networks[networkName].gatekeepers`.

```
         // Remove gatekeeper
         for(uint i = 0; i < currentGatekeepers.length; i++) {
             if(currentGatekeepers[i] == gatekeeper) {
                 // Swap gatekeeper to be removed with last element in the array
-                _networks[networkName].gatekeepers[i] =
_networks[networkName].gatekeepers[currentGatekeepers.length - 1];
+                currentGatekeepers[i] = currentGatekeepers[currentGatekeepers.length
- 1];
                 // Remove last element in array
-                _networks[networkName].gatekeepers.pop();
+                currentGatekeepers.pop();
             }
```

12. It would be safer to call `FlexibleNonceForwarder._refundExcessValue()` at the end of the function `FlexibleNonceForwarder.execute()`.

Identity: Acknowledged.

Renascence: This issue has been acknowledged.

[I-7] `ForwardRequest` should contain a deadline parameter

Context:

- IForwarder.sol#L7-L14

Description: `IForwarder.ForwardRequest` doesn't have a deadline parameter, so it's possible for a request to be executed much later than when it was submitted.

Recommendation: A deadline parameter can be added similar to OpenZeppelin's implementation.

Identity: Acknowledged.

Renascence: This issue has been acknowledged.