FYEO

Theoriq

Repo: https://github.com/chain-ml/theoriq-smart-contracts

Security Review Update: December 4, 2025

Reviewer: balthasar@gofyeo.com

Security Code Review Theoriq

New security issues, 3

After the development team implemented the latest updates, FYEO conducted a review of the modifications. The primary goal of this evaluation was to ensure the continued robustness of the program's security features, safeguarding user funds and maintaining the overall integrity of the program.

General Updates:

The update introduces LayerZero cross-chain token functionality, enabling Theoriq tokens to move seamlessly between Ethereum, Base, etc. At the core of this expansion is a new base contract that provides shared security and administrative controls for all cross-chain operations. This foundation implements strict role-based permissions, emergency pause capabilities, and dedicated LayerZero administrative roles. Building on this base, two specialized contracts handle different deployment scenarios: one creates and manages tokens on chains where they don’t natively exist by minting and burning as needed, while the other wraps the existing Ethereum token to make it bridge-compatible by locking and unlocking the original tokens. Both contracts integrate with a registry system to enforce user restrictions across all chains, ensuring banned users cannot circumvent controls by moving between networks. The entire system uses isolated storage layouts designed for safe upgrades, explicit initialization processes that reduce reliance on inherited defaults, and consistent security patterns across all components.

Separately, the registry system receives a more robust and upgrade-friendly redesign to support the expanded multi-chain environment. A new version introduces logic that prevents restricted users from escaping their status through self-removal, which is particularly important now that users can interact across multiple blockchains. Core validation logic is extracted into a dedicated base layer so that future extensions can evolve cleanly while keeping essential behavior consistent across all chains. The result is a clearer separation of responsibilities, improved safety around administrative operations, and a more resilient structure for managing user restrictions that works uniformly whether users are on Ethereum, Base, BSC, or moving tokens between them.

RISK OF PERMANENTLY-LOCKED ROLE MANAGEMENT

Finding ID: FYEO-THEORIQ-01
Severity: **Low**
Status: **Remediated**

Description

Multiple related issues around ownership, role management and pauseability in `BaseTheoriqLayerZeroContract` create inconsistent admin semantics and a risk of permanently locking critical administrative flows while paused. The problems are:

- `renounceRole` is not guarded by pause, while `grantRole`/`revokeRole` are - inconsistent RBAC pause policy.
- `OwnableUpgradeable` is left uninitialized, do document this
- `_checkOwner()` is guarded by `whenNotPaused`, so `onlyOwner` (LayerZero admin) calls are blocked when paused, may prevent recovery actions in emergencies.
- The contract mixes AccessControl (DEFAULT_ADMIN_ROLE + LZ_ADMIN_ROLE) and Ownable, but `onlyOwner` effectively checks `LZ_ADMIN_ROLE` (not DEFAULT_ADMIN_ROLE) while `owner()` returns the DEFAULT_ADMIN - confusing and error-prone.
- Because role management (`grantRole`/`revokeRole`) is blocked while paused, a paused contract may be irrecoverable if pauser keys are lost or compromised.

Proof of Issue

File name: layerzero-oapp/contracts/BaseTheoriqLayerZeroContract.sol

```
abstract contract BaseTheoriqLayerZeroContract is
    ...
    AccessControlDefaultAdminRulesUpgradeable,
    OwnableUpgradeable

// Ownable ownership transfer disabled
function _transferOwnership(address) internal virtual override {
    revert OwnableOwnershipChangeDisabled();
}

// owner semantics mixed with roles, _checkOwner blocks when paused
function _checkOwner() internal view virtual override whenNotPaused {
    if (!hasRole(LZ_ADMIN_ROLE, _msgSender())) {
        revert OwnableUnauthorizedAccount(_msgSender());
    }
}

// Role management blocked when paused, but renounceRole not overridden
function grantRole(bytes32 role, address account) public override whenNotPaused
{
    AccessControlDefaultAdminRulesUpgradeable.grantRole(role, account);
}
function revokeRole(bytes32 role, address account) public override
whenNotPaused {
```

```
    AccessControlDefaultAdminRulesUpgradeable.revokeRole(role, account);
}
```

Severity and Impact Summary

A paused contract can become **irrecoverable** if the pauser loses their key or a malicious pauser both pauses and ensures no pauser/admin can unpause (by revoking or renouncing roles). Blocking role management while paused removes the normal recovery path.

Mixing AccessControl and Ownable semantics in a non-symmetric way (owner() <> DEFAULT_ADMIN, onlyOwner <> LZ_ADMIN_ROLE) is error-prone and may break LayerZero expectations.

`renounceRole` being callable while paused is inconsistent with `grantRole`, `revokeRole` behavior and could be used (intentionally or accidentally) to remove critical roles during an enforced pause.

Leaving Ownable uninitialized is acceptable if intentional and documented, otherwise it can surprise integrators or inherited code.

Recommendation

Make pause policy consistent for role management. Decide and document single source of truth for admin semantics. Re-evaluate `_checkOwner()` pause semantics. Avoid zero-address initial role assignments. Add tests & deploy-time guards.

Inconsistent and Incomplete Deployment Configuration

Finding ID: FYEO-THEORIQ-02
Severity: **Informational**
Status: **Remediated**

Description

Several inconsistencies and omissions have been identified in the deployment configuration related to network endpoint definitions, chain parameter completeness, null registry values, and inconsistent assignment of privileged roles (owner/admin). These issues can lead to misconfigured deployments, incorrect network targeting, and unexpected assignment of contract ownership.

This combined issue outlines four related concerns that all stem from deployment configuration problems.

Proof of Issue

File name: layerzero-oapp/hardhat.config.ts
Line number: 80

```
sepolia: {
    eid: EndpointId.SEPOLIA_V2_TESTNET,
    url: process.env.RPC_URL_BASE_MAINNET ||
'https://ethereum-sepolia-rpc.publicnode.com',
    accounts,
},
ethereum: {
    eid: EndpointId.ETHEREUM_V2_MAINNET,
    url: process.env.RPC_URL_ETHEREUM ||
'https://ethereum-sepolia-rpc.publicnode.com',
    accounts,
},
```

`sepolia` defaults to a **BASE mainnet var**, and `ethereum` falls back to a **Sepolia URL**, which appears inverted.

File name: layerzero-oapp/deploy/params/TheoriqOFT.params.ts
Line number: 32

```
export const chainParameters: Partial<Record<EndpointId,
Partial<TheoriqOFTParameters>>> = {
    // BASE configuration missing entirely
};
```

BASE networks appear absent despite being referenced elsewhere.

File name: layerzero-oapp/deploy/params/TheoriqOFT.params.ts
Line number: 43

```
[EndpointId.BSC_V2_MAINNET]: {
    ...productionParameters,
    registry: null,
},
```

A `null` registry on mainnet deployments is likely unintended.

File name: layerzero-oapp/deploy/params/TheoriqOFT.params.ts
Line number: 15

```
export const createDefaultParameters = (deployer: string): TheoriqOFTParameters
=> ({
    registry: null,
    defaultAdmin: deployer,
    pauser: deployer,
    lzAdmin: deployer,
    lzDelegate: deployer,
    upgrader: deployer,
    ...
```

The deployment script automatically assigns the deployer as the owner and default admin across networks, which may not align with expected governance/ops structure.

Severity and Impact Summary

Collectively, these indicate systemic configuration drift and could cause deployment failures or unsafe ownership states.

Recommendation

Validate and correct all RPC URL fallbacks to ensure network alignment. Ensure production deployments never default to `registry: null`. Avoid setting the deployer as owner/admin by default.

Token & endpoint passed to implementation constructor

Finding ID: FYEO-THEORIQ-03
Severity: **Informational**
Status: **Acknowledged**

Description

The TheoriqOFTAdapter inherits from LayerZero's OFTAdapterUpgradeable, which defines the underlying token address as an immutable value set in the implementation contract's constructor. Because immutable variables live in the implementation's bytecode and not in proxy storage, the token address becomes permanently bound at implementation deployment. As a result, any proxies pointing to the same implementation would all be locked to the same token, and this binding cannot be changed through proxy initialization.

Currently, the codebase avoids this issue by deploying a dedicated implementation for each proxy. However, this pattern is fragile and non-standard, and could break if someone later attempts to "optimize" by reusing a single implementation across multiple proxies.

Proof of Issue

File name: layerzero-oapp/contracts/TheoriqOFTAdapter.sol

```
constructor(address _token, address _lzEndpoint)
    OFTAdapterUpgradeable(_token, _lzEndpoint)
{
    _disableInitializers();
}

// Proxy initializer does not forward token/_lzEndpoint to OFTAdapter's
initialize
function __TheoriqOFTAdapter_init(... ) internal onlyInitializing {
    __BaseTheoriqLayerZeroContract_init(...);
    __OFTAdapter_init(lzDelegate);
}
```

Severity and Impact Summary

Pattern works correctly but is non-standard (most upgradeable contracts share one implementation), fragile (easy to accidentally "optimize" incorrectly) and poorly documented (risk of future mistakes)

Recommendation

The current implementation is functionally correct but uses a non-standard and fragile pattern. The risk is with future modifications. Adding documentation and validation will prevent this class of errors. Add runtime validation in initialize(). Add deployment script validation. Add test coverage for the pattern.

Commit Hash Reference:

For transparency and reference, the security review was conducted on the specific commit hash for the Theoriq repository. The commit hash for the reviewed versions is as follows:

ddb1165c03a0815c110989be6293cb726671cbcb

Remediations were submitted with the commit hash 9984816edcb0405eea6ae785562ad9c0b1cae769.

Conclusion:

In conclusion, the security aspects of the Theoriq program remain robust and unaffected by the recent updates. Users can confidently interact with the protocol, assured that their funds are well-protected. The commitment to security exhibited by the development team is commendable, and we appreciate the ongoing efforts to prioritize the safeguarding of user assets.