OpenZeppelin | security

CACEIS EURXT Smart Contract Security Audit

caceis
INVESTOR SERVICES

May 22, 2026

This document has been prepared solely for the named client and it may not be relied upon by any third parties. The content in this document is provided for informational purposes only and does not constitute professional advice of any kind. Security assessments are limited by factors such as scope, time, and techniques used and therefore this document does not constitute a certification, guarantee, or warranty regarding the proper functioning of any system or the absence of security vulnerabilities.

Copyright © 2026 Zeppelin Group Ltd.

Table of Contents

Table of Contents 2

Summary 3

Executive Summary 4

Scope 5

System Overview 6

Security Model and Trust Assumptions 7

Additional Considerations 9

Low Severity 11

L-01 Maintenance Functions Gated by DEFAULT_ADMIN_ROLE Increase Admin Key Exposure 11

L-02 Tokens Can Be Minted Before Redemption Address Is Set 11

L-03 seizeBlacklistedFunds Allows Seized Funds to Be Sent to the Redemption Address 12

Notes & Additional Information 13

N-01 Orphaned NatSpec Block 13

N-02 Storage __gap Reserves 49 Slots Instead of the Conventional 50 13

N-03 __AccessControlEnumerable_init Not Invoked in initialize 13

N-04 Custom Errors in require Statements 14

N-05 Missing Security Contact 14

N-06 Missing Named Parameters in Mapping 14

N-07 Unused EmptyString Error in LibErrors 15

N-08 ContractURIUpdated Event Signature Diverges from ERC-7572 15

N-09 CannotRenounceLastRole Reused by revokeRole 15

N-10 Last-Member Guard on All Roles Restricts Incident Response 15

N-11 Last-Member Guard Duplicated Between revokeRole and renounceRole 16

N-12 Inaccurate Docstrings 17

N-13 Deviations from the Solidity Style Guide 17

N-14 Use of OpenZeppelin Contracts v4.9.x Requires Legacy __gap Storage Pattern 17

N-15 Redundant Code 18

Conclusion 20

Appendix 22

File-level Hashes 22

Issue Classification 22

Summary

Type	Stablecoins
Timeline	From 2026-05-20 To 2026-05-21
Languages	Solidity

Total Issues	3 (0 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	0 (0 resolved)
Low Severity Issues	3 (0 resolved)
Notes & Additional Information	15 (0 resolved)
Client Reported Issues	0 (0 resolved)

Executive Summary

OpenZeppelin was engaged by CACEIS to conduct a smart contract security audit of the `EURXT` contract, an upgradeable single-issuer euro stablecoin being designed for issuance under the MiCA framework. The goal of this engagement was to identify potential security vulnerabilities, verify the robustness of the business logic, and ensure the system aligns with best practices for permissioned tokenization architectures. The review was strictly limited to production-intended smart contracts. No mock contracts, unit tests, deployment scripts, or proxy initialization scripts were included in the scope. The Transparent Proxy and proxy administrator that govern upgrades, the multi-signature wallets and off-chain governance procedures that manage privileged role assignment, the trusted forwarder contract operated through the Taurus custody platform, and the off-chain settlement infrastructure responsible for redemption attribution and burn triggering were also outside the review boundary.

The smart contract security audit did not identify any critical, high, or medium severity issues. Three low severity issues were raised, all of which describe operational refinements to the design rather than exploitable security defects. The three findings concern minimizing routine use of the most privileged role, preventing seized funds from being routed directly to the redemption address, and preventing token issuance before the redemption address has been configured. None of these findings describe a path that an external party can exploit against the contract or its holders. Each requires an already-trusted privileged operator to perform an action that the CACEIS operations manual explicitly excludes from normal procedure, and each is recoverable through ordinary administrative actions should it occur. As a result, even if these three findings remain in their current form at launch, they do not introduce material risk to token holders, to the issuer's reserves, or to the contract's regulatory posture. The smart contract security audit did not identify any defect that would allow unauthorized minting, unauthorized burning, theft of holder balances, evasion of the pause, evasion of the blacklist, or compromise of the role-based access control model.

CACEIS has acknowledged the three low severity findings and has decided to address them in the next version of the smart contract, which will be the version deployed at launch.

Scope

OpenZeppelin performed an audit of the implementation of the `EURXT` stablecoin contract. The codebase was provided as a ZIP file with SHA-256 hash `345e7f3ca1b6d2d726650af4baaf2ec3eae2df4468a7cf732b6a2920c7633951`. Individual file-level hashes are listed in the Appendix.

In scope were the following files:

```
src
├── EURXT.sol
└── library
    ├── LibErrors.sol
    └── LibModifiers.sol
```

None of the findings raised in this report have been remediated at this time. CACEIS has provided the following statement:

> CACEIS acknowledges the three low findings. After careful analysis, we have decided to include the necessary fixes in the next version of the smart contract and proceed to go live with that release.

System Overview

`EURXT` (the Euro eXchange Token) is a single-issuer euro stablecoin contract from CACEIS. The contract represents one euro of issuer liability per token, uses six decimal places, and is designed to be deployed once as a Transparent Proxy implementation that delegates to a single `EURXT` logic contract. CACEIS has been authorized by the French ACPR under MiCA for crypto-asset services, and `EURXT` is being designed as a MiCA-regulated electronic-money token (EMT) under that regime rather than as a permissionless stablecoin. The deployment is pre-launch at the time of review, so the recommendations in this report are made without backwards-compatibility constraints on storage layout or external interfaces.

The contract is built on the OpenZeppelin v4.9.4 upgradeable base contracts. It exposes the standard ERC-20 token surface, supports EIP-2612 permit-based approvals, and supports EIP-2771 meta-transactions to enable gasless approval and transfer flows through a trusted forwarder. An emergency pause mechanism can halt all token movement, and role-based access control is used to gate every privileged operation. Role administration is further restricted on-chain to ensure that no role can be unintentionally lost, by always requiring at least one holder to remain.

Issuance and redemption are intentionally asymmetric. Minting transfers freshly issued tokens to a recipient nominated by the minter, modeling the on-ramp side of the off-chain fiat flow. Burning is restricted to a single redemption address configured by the administrator, modeling the off-ramp side: token holders redeem by sending `EURXT` to the redemption address, where the off-chain settlement infrastructure observes the inflow, settles a fiat payout, and then triggers the burn to destroy the staged tokens. The redemption address can be rotated by the administrator and is not constrained at the contract level to be empty before rotation. Regulatory seizure is implemented as a paired operation: an address is first added to the blacklist, then a dedicated seizure path moves the flagged balance to a compliant destination through a controlled bypass of the sender-blacklist guard.

Every token movement passes through a central transfer hook that enforces the pause state, the sender-blacklist check (bypassed during regulatory seizure, and not applicable to mints since mints have no sender), and the recipient-blacklist check, which is enforced on every transfer including mints. All standard transfer paths route through this hook, including direct transfers, allowance-based transfers, and meta-transactions relayed by the trusted forwarder. As a result, no transfer path permits a blacklisted address to send tokens, and no transfer path

permits the contract to credit a blacklisted recipient outside the explicit seizure flow. Allowance creation, whether through direct approval calls or EIP-2612 permits, is not routed through this hook, so allowances can be granted while the contract is paused or by or for a blacklisted address. Only the subsequent transfer is gated. The blacklist exception for allowances is documented in the contract, and the equivalent pause exception is the standard OpenZeppelin behavior and is inherited unchanged.

Security Model and Trust Assumptions

The on-chain access control surface defines six roles. All role-holders are assumed to be honest and to follow the off-chain governance procedures described in the CACEIS operations and access-control policy documents. Findings in this report do not flag scenarios that simply require multiple trusted role-holders to collude, unless such collusion breaks an explicit invariant.

`DEFAULT_ADMIN_ROLE`: Acts as the administrator of all other roles. May grant, revoke, and renounce roles, rotate the redemption address, and update the on-chain metadata URI. Granted at initialization; all other roles are granted post-deployment.

The administrator is trusted to follow the off-chain governance procedures documented in the CACEIS access-control policy when managing roles. Specifically, the following role-management trust assumptions are not enforced on-chain and rely on the administrator's discipline:

- A role is never granted to an address that is currently blacklisted. The on-chain layer does not cross-check the blacklist when granting roles, so an inadvertent grant could place an operational role at an address that is unable to use it.
- A role is never granted to an address known to be malicious or otherwise unsuitable, including OFAC-sanctioned addresses, addresses associated with confirmed exploits, and addresses flagged by other applicable sanctions or compliance lists. The on-chain layer performs no such cross-check.
- A role is never granted to the trusted forwarder address. Role checks resolve the caller via the EIP-2771 sender-resolution path, which can return the forwarder address itself on direct calls from the forwarder when the calldata is shorter than the appended-sender

suffix. A forwarder that holds a role could therefore exercise that role through such short-calldata calls.

- A role is never granted to the contract itself, the zero address, or any other dead or unrecoverable address. The on-chain layer permits such grants, which would result in a role-holder that cannot sign transactions.
- All role grants and revocations follow the cross-team quorum requirements defined in the off-chain governance policy. The on-chain role-administration functions enforce only that the caller holds the role-admin for the target role; any further multi-party approval is the responsibility of the custody platform.

`MINTER_ROLE`: May mint new tokens to any non-zero, non-blacklisted recipient. The minter is assumed to mint only when the corresponding fiat funds have been received and reserved off-chain, and never to mint before the redemption address has been configured, since redemptions cannot be processed until that configuration is in place. The minter is further assumed not to mint directly to the redemption address, which would inject tokens into the off-chain redemption attribution flow without a corresponding redemption ticket, and not to mint to the `EURXT` contract address itself, since the on-chain rescue path rejects `EURXT` as the token argument and any balance accruing at the contract address is therefore permanently stranded.

`BURNER_ROLE`: May destroy tokens held at the redemption address; burns are not permitted from any other source. The burner is assumed to burn tokens only after the off-chain settlement system has confirmed that the corresponding fiat payout to the redeemer has been completed.

`PAUSER_ROLE`: May pause and resume all token movement, including seizure operations; allowance creation is not affected by the pause. The pauser is assumed to use this capability only as an emergency circuit-breaker, and to recognize that allowances may continue to accumulate during pause windows and become spendable as soon as the pause is lifted.

`BLACKLIST_ADMIN_ROLE`: May add or remove addresses from the blacklist, and may forcibly transfer the balance of a blacklisted address to a non-blacklisted destination through the on-chain seizure flow. The blacklist gates only token movement and does not block allowance creation, so any unsettled allowance involving a blacklisted party remains in storage but reverts at settlement time. The blacklist admin is assumed to operate under a documented compliance procedure aligned with MiCA freeze-and-seize requirements, and to direct seizures only to compliant destinations.

`RESCUER_ROLE`: May rescue arbitrary foreign ERC-20 tokens held by the contract to any non-zero destination. The rescue path explicitly rejects `EURXT` itself as the token argument, which prevents the rescuer from sweeping `EURXT` accidentally sent to the contract by users. The

rescuer is assumed to direct rescues to legitimate recovery destinations consistent with the original sender's intent.

Additional Considerations

Trusted forwarder (EIP-2771): The contract supports EIP-2771 meta-transactions to enable gas-sponsored execution of operator transactions through the Taurus custody platform's fee-payer functionality. The trusted forwarder address is fixed in the implementation bytecode at deployment rather than stored, so rotation of the forwarder requires deploying a new implementation and upgrading the proxy. CACEIS has confirmed that the forwarder is restricted to internal Taurus/CACEIS-controlled addresses and is not exposed to public or third-party relayers, and that the forwarder contract itself is derived from the OpenZeppelin reference implementation, which verifies the signature of the original sender against the relayed call before appending the sender's address to the calldata suffix. The forwarder therefore cannot fabricate calls on behalf of role-holders; it can only relay transactions already signed by an account that holds the relevant role for the targeted privileged function, and the standard access-control checks continue to gate on the resolved original sender. The forwarder operator is held to the same trust assumptions as the privileged role-holders whose meta-transactions it relays, and the forwarder is assumed to remain restricted to the internal Taurus/CACEIS operator set. Compromise of the forwarder operator key does not on its own extend privileged-action capability beyond what the existing administrative trust model already implies, since the underlying access control still requires the original signer to hold the relevant role.

Off-chain settlement infrastructure: The off-chain redemption pipeline observes inflows to the redemption address and is responsible for matching them to redemption tickets, settling fiat payouts, and instructing the burner to destroy the staged tokens. The contract does not encode this attribution: any transfer to the redemption address is indistinguishable on-chain from any other. Operators are trusted not to co-mingle seized funds, recovered funds, or other non-redemption flows into the redemption address, and the off-chain pipeline is trusted to correctly attribute inflows to legitimate redemption tickets. The review of the on-chain contract does not extend to the off-chain components that mint, settle redemptions, or trigger burns.

Proxy and upgrade authority: The contract is intended to be deployed behind a Transparent Upgradeable Proxy managed by an external proxy administrator. Deployment scripts, the proxy itself, the proxy administrator configuration, and the upgrade authority are out of scope for this engagement and cannot be assessed from the source alone. Holders of the proxy administrator key may upgrade `EURXT` to arbitrary logic, including logic that rewrites balances or removes operational restrictions, and are trusted accordingly.

Test coverage and deployment scripts: No automated test suite, deployment scripts, or proxy initialization scripts were provided as part of the in-scope material. Coverage of initialization correctness, role-granting workflows, and post-deployment operational checks therefore relies on the procedures documented in the CACEIS operations manual rather than on observed code. The internal test suite that CACEIS maintains around the contract was not included in scope, so the audit was unable to directly observe its coverage. The following areas are flagged as worth confirming that they are explicitly covered in the existing suite ahead of deployment: exhaustive role-gating tests that confirm each privileged actor can only invoke the functions intended for that role and reverts on every other privileged entry point, branch coverage of every revert path in the contract and its libraries, full lifecycle coverage of the mint, transfer, blacklist, seize, burn, pause, and upgrade flows, and integration tests that exercise the trusted forwarder relay against each user-facing entry point. Without such coverage, regressions introduced during future upgrades may silently expand the surface available to any given role and would not be detected by source-level audit alone.

Low Severity

L-01 Maintenance Functions Gated by `DEFAULT_ADMIN_ROLE` Increase Admin Key Exposure

In OpenZeppelin's `AccessControl` pattern, `DEFAULT_ADMIN_ROLE` is the role used to grant and revoke every other role in a contract, and is intended to be reserved for role administration. In `EURXT`, this role is also used to gate two maintenance operations: `setRedemptionAddress`, which updates the address from which tokens may be burned during redemption, and `setContractURI`, which updates the off-chain metadata URI. Both functions are guarded by `onlyRole(DEFAULT_ADMIN_ROLE)`, requiring the same key that controls role administration to be used for routine state changes.

This deviates from the principle of least privilege. Because `DEFAULT_ADMIN_ROLE` controls role administration for the entire contract, increasing the frequency with which the holding key must sign transactions also increases its exposure to operational mistakes, signer phishing, and hardware-wallet blind-signing attacks. A dedicated lower-privilege role for these actions would allow the admin key to remain cold and limit any single signing-time compromise to non-administrative state.

Consider introducing one or more lower-privilege roles to gate `setContractURI` and `setRedemptionAddress`, and restricting `DEFAULT_ADMIN_ROLE` to role-administration duties.

L-02 Tokens Can Be Minted Before Redemption Address Is Set

After `initialize`, `_redemptionAddress` defaults to `address(0)`. In this state `burn` reverts unconditionally because it requires `from == _redemptionAddress` while `LibModifiers.checkNonZero(from)` simultaneously forbids `from == address(0)`. The `mint` function has no equivalent precondition, so an account holding `MINTER_ROLE` can issue tokens before `setRedemptionAddress` has been called. Any tokens minted in that window have no on-chain redemption path until the redemption address is configured, even

though the operational lifecycle requires a redemption address to be in place before minting begins.

Consider adding a guard in `mint` that reverts when `_redemptionAddress == address(0)`, enforcing the deployment ordering on-chain so that tokens cannot be issued ahead of a configured redemption address. Alternatively, consider accepting the initial redemption address as a parameter of `initialize` so the window cannot exist.

L-03 `seizeBlacklistedFunds` Allows Seized Funds to Be Sent to the Redemption Address

`seizeBlacklistedFunds` transfers tokens from a blacklisted `from` address to an arbitrary `to` address. The only restrictions placed on `to` are that it is non-zero and not blacklisted. In particular, there is no check that `to` is not equal to `_redemptionAddress`.

`_redemptionAddress` is the staging address for the redemption flow: users send `EURXT` to it, and the off-chain redemption infrastructure observes those inflows to settle fiat payouts and eventually trigger `burn` via `BURNER_ROLE`. Sending seized funds directly to `_redemptionAddress` mixes them into the redemption accounting and, depending on how the off-chain system attributes inflows, could trigger an unintended fiat payout or otherwise corrupt redemption bookkeeping. Even if the off-chain system requires an explicit redemption ticket per inflow, co-mingling seized balances with voluntary redemptions adds operational risk that the contract can prevent at the on-chain layer.

Consider rejecting `to == _redemptionAddress` in `seizeBlacklistedFunds`. If a regulatory order requires destruction of seized funds, the operator can move them first to a treasury address controlled by `BURNER_ROLE`, or any other intermediate destination, and then transition them into the redemption flow through a separate, deliberate step rather than as a side effect of seizure.

Notes & Additional Information

N-01 Orphaned NatSpec Block

`EURXT.sol` contains a NatSpec block at lines 424 to 430 that documents a `burn` function, but is followed by the `Storage` section header comment with no function declaration beneath it. The block appears to be an artifact of a prior refactor or code move that left the documentation detached from any code.

Consider removing the orphaned NatSpec block so that all documentation is colocated with the code it describes.

N-02 Storage `__gap` Reserves 49 Slots Instead of the Conventional 50

`EURXT` declares four state variables before its storage gap, but Solidity packs `_seizureInProgress` (a `bool`) and `_redemptionAddress` (an `address`) into a single slot because they are declared consecutively and fit together in 32 bytes. The four variables therefore occupy three slots, and combined with `__gap[46]` the contract reserves 49 slots rather than the 50 that the OpenZeppelin upgradeable storage convention reserves per contract.

Consider increasing the gap to `__gap[47]` so the storage region matches the conventional 50-slot reservation.

N-03 `__AccessControlEnumerable_init` Not Invoked in `initialize`

In `EURXT.sol`, the `initialize` function calls `__AccessControl_init` but does not call `__AccessControlEnumerable_init` from the inherited `AccessControlEnumerableUpgradeable`. In OpenZeppelin Contracts v4.9.x this initializer performs no setup, so the omission is currently safe. If a future version of OpenZeppelin Contracts adds setup work to `__AccessControlEnumerable_init`, that setup would be silently skipped on any subsequent upgrade of the implementation to that version.

Consider invoking `__AccessControlEnumerable_init` immediately after `__AccessControl_init` in `initialize`.

N-04 Custom Errors in `require` Statements

Since Solidity `0.8.26`, custom errors can be used inside `require` statements. Initial support was limited to the IR pipeline. Solidity `0.8.27` extended this to the legacy pipeline as well.

Throughout the codebase, every `if (...) revert ...` pattern could equivalently be expressed as a `require` statement that reverts with the same custom error.

Consider replacing the `if-revert` patterns with equivalent `require` statements using the same custom errors, for conciseness and a small gas saving.

N-05 Missing Security Contact

Providing a security contact (such as an email address or ENS name) within a smart contract simplifies communication when a vulnerability is identified. It lets the contract owners specify the disclosure channel, reducing the risk that a reporter fails to surface an issue because they do not know where to send it. It also gives upstream maintainers a direct path to notify the owners about bugs found in third-party libraries used by the contract.

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

N-06 Missing Named Parameters in Mapping

Since Solidity 0.8.18, mappings can include named parameters in the form `mapping(KeyType KeyName? => ValueType ValueName?)` to clarify the role of the key and the value.

The `_blacklisted` mapping in `EURXT` declares neither a key name nor a value name.

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

N-07 Unused `EmptyString` Error in `LibErrors`

The `EmptyString` error is declared in `LibErrors` but is never reverted from anywhere in the codebase.

Consider removing the `EmptyString` error declaration.

N-08 `ContractURIUpdated` Event Signature Diverges from ERC-7572

The `setContractURI` function emits `ContractURIUpdated(string newURI)`, while the canonical ERC-7572 event signature is `ContractURIUpdated()` with no parameters. The two signatures hash to different `topic[0]` values, so off-chain indexers, block explorers, and metadata refresh services that subscribe to the standard ERC-7572 event will not detect updates emitted by this contract.

Consider changing the event declaration to `event ContractURIUpdated()` and removing the `newURI` argument from the `emit`. Consumers that need the new URI value can read it via `contractURI()` on the next block, as ERC-7572 intends.

N-09 `CannotRenounceLastRole` Reused by `revokeRole`

The `revokeRole` function reverts with `LibErrors.CannotRenounceLastRole(role)` when the caller is removing the last holder of a role. The name reads as a contradiction at this call site, since the caller is revoking another account's role rather than renouncing their own.

Consider renaming the error to a neutral form such as `CannotRemoveLastRoleHolder` so that it reads correctly from both `renounceRole` and `revokeRole`.

N-10 Last-Member Guard on All Roles Restricts Incident Response

The `renounceRole` and `revokeRole` overrides apply a last-member guard to all six roles defined by `EURXT`: `DEFAULT_ADMIN_ROLE`, `MINTER_ROLE`, `BURNER_ROLE`, `PAUSER_ROLE`, `BLACKLIST_ADMIN_ROLE`, and `RESCUER_ROLE`. When

`getRoleMemberCount(role) <= 1`, both functions revert with `CannotRenounceLastRole`.

For `DEFAULT_ADMIN_ROLE` this reflects a real risk, since losing the sole admin would permanently disable role administration. For the five operational roles, `DEFAULT_ADMIN_ROLE` can always re-grant them, so reaching a zero-holder state would be recoverable. The uniform guard, however, makes that zero-holder state unreachable: an incident response procedure that needs to temporarily revoke all holders of an operational role (for example, suspending every `MINTER_ROLE` holder while investigating a suspected key compromise) cannot be executed at all. The last holder cannot be revoked, and granting a placeholder beforehand does not help, since revoking the placeholder would itself trip the guard.

The OpenZeppelin library provides `AccessControlDefaultAdminRules`, which protects only `DEFAULT_ADMIN_ROLE` and does so more strongly than a count-based check: it enforces a two-step transfer with a configurable delay, requires renunciation to follow the same delayed path, and ensures there is exactly one admin at a time. Operational roles remain under standard `AccessControl` semantics and stay fully revocable by an authorised admin.

Consider adopting `AccessControlDefaultAdminRules` for `DEFAULT_ADMIN_ROLE` and removing the last-member guard from the five operational roles. This would preserve strong protection of `DEFAULT_ADMIN_ROLE` while restoring the ability to temporarily zero out an operational role during incident response.

N-11 Last-Member Guard Duplicated Between `revokeRole` and `renounceRole`

`revokeRole` and `renounceRole` both apply the same last-member guard before delegating to `super`. Since both functions route through the internal `_revokeRole` hook in `AccessControlUpgradeable`, the guard could instead be placed in an override of `_revokeRole`, applying it to both entry points from a single location and removing the duplicated logic as well as the multi-base override declarations.

Consider moving the last-member guard into an override of `_revokeRole` and removing the duplicated checks from `revokeRole` and `renounceRole`.

N-12 Inaccurate Docstrings

Throughout the codebase, several docstrings do not match the implementation:

- The storage-layout comments in the contract docstring and at the `__gap` declaration in `EURXT.sol` state that three custom storage variables are declared and enumerate `_blacklisted`, `_contractURI`, and `_seizureInProgress`. These comments omit `_redemptionAddress` and the fact that `_seizureInProgress` and `_redemptionAddress` are packed into a single storage slot.
- The `@dev` block on `isBlacklisted` describes the function as being used "on-chain by mint/burn guards and `_beforeTokenTransfer`". In practice, those on-chain paths read the `_blacklisted` mapping directly, so the function is only used off-chain.

Consider updating each of the docstrings listed above so that they accurately describe the current implementation.

N-13 Deviations from the Solidity Style Guide

`EURXT.sol` contains deviations from the Solidity style guide conventions. One instance is the declaration of the `RedemptionAddressUpdated` event at line 444, between function definitions, while all other events (`Blacklisted`, `UnBlacklisted`, `BlacklistedFundsSeized`, `ContractURIUpdated`, `RescueERC20`) are grouped together at lines 253 to 277. The style guide recommends grouping event declarations near the top of the contract rather than interleaving them with functions.

Consider relocating `RedemptionAddressUpdated` to the events section alongside the other event declarations, and reviewing the contract for any other deviations from the Solidity style guide.

N-14 Use of OpenZeppelin Contracts v4.9.x Requires Legacy `__gap` Storage Pattern

The `EURXT` token is deployed behind a `TransparentUpgradeableProxy` and inherits from `v4.9.4` of the OpenZeppelin Contracts Upgradeable library. This release predates v5.x, which migrates upgradeable contracts to the ERC-7201 namespaced storage layout. As a result, `EURXT` relies on the legacy `uint256[50]` reserved-slots convention, in which every contract in the inheritance chain reserves a fixed-size `__gap` array and must decrement that array in

step with any new state introduced by a parent. Any drift in this accounting silently corrupts storage on upgrade.

The cost of this convention is not limited to keeping a counter aligned. Because each parent's storage is laid out sequentially with the child's, any future change to the inheritance chain (for example, introducing a new base contract, removing one, reordering inheritance, or adopting a parent that itself adds state) shifts every downstream variable and requires a coordinated rewrite of `__gap` declarations across the chain to preserve the existing storage layout. In a deployed, upgradeable contract this is a high-risk operation: a single miscalculation produces silent storage collisions whose effects can range from subtle accounting bugs to full loss of contract state. ERC-7201 namespaced storage, adopted in v5.x, eliminates this class of problem by giving each contract a derived storage slot that is independent of its position in the inheritance chain, so adding, removing, or reordering parents no longer affects the layout of any other contract.

Consider migrating to OpenZeppelin Contracts v5.x, which adopts ERC-7201 namespaced storage and removes the cross-contract `__gap` bookkeeping required by the v4.x inheritance model. Performing this migration before deployment makes future upgrades materially less risky and removes a recurring source of upgrade hazards.

N-15 Redundant Code

Throughout the codebase, several code redundancies were identified that increase gas costs and maintenance surface. In particular:

- In `mint`, the recipient is checked against `_blacklisted` before calling `_mint`. The same check is then performed inside `_beforeTokenTransfer`, which `_mint` invokes, and both paths revert with `LibErrors.Blacklisted(to)`.
- `rescueERC20` reverts with a dedicated `LibErrors.RescueAmountZero` for the zero-amount check, while every other zero-amount check in the contract goes through `LibModifiers.checkNonZeroAmount` and reverts with the generic `LibErrors.ZeroAmount`. The dedicated error can be removed and the check unified with `LibModifiers.checkNonZeroAmount`.
- In `seizeBlacklistedFunds`, the `_blacklisted[to]` check is redundant: `_beforeTokenTransfer` performs the same recipient-blacklist check unconditionally (the `_seizureInProgress` guard only skips the sender check) and reverts with the same `LibErrors.Blacklisted(to)` error. The `LibModifiers.checkNonZero(to)` call is also redundant with the zero-address check inside `ERC20Upgradeable._transfer`, although removing it would change

the revert from `LibErrors.ZeroAddress` to the OpenZeppelin default string, so retaining the explicit check for error-message consistency is also a reasonable choice.

- In `supportsInterface`, the explicit check against `type(IAccessControlUpgradeable).interfaceId` is redundant: `AccessControlUpgradeable.supportsInterface`, reached via the `super.supportsInterface` fallback, already returns `true` for that interface ID.
- In `_beforeTokenTransfer`, the call to `LibModifiers.checkNotPaused(paused())` duplicates the work of `PausableUpgradeable._requireNotPaused()` (or equivalently the `whenNotPaused` modifier). Removing the wrapper would change the revert from `LibErrors.TokenPaused` to the OpenZeppelin default string `"Pausable: paused"`, so retaining the explicit check for error-message consistency is also a reasonable choice.
- The literal `6` representing the token's decimal precision is duplicated: it is returned directly by `decimals()` and used again as the exponent in `MAX_SUPPLY = 5_000_000_000 * 10 ** 6`. Defining a single private constant such as `uint8 private constant DECIMALS = 6` and referencing it from both sites would remove the duplication and keep the two values in sync.

Consider removing the redundant code and consolidating the checks and constants listed above where doing so does not conflict with explicit error-message requirements.

Conclusion

`EURXT` implements a single-issuer, pre-launch euro stablecoin intended for issuance by CACEIS as a MiCA-regulated electronic-money token under its ACPR authorization. OpenZeppelin audited the implementation of the `EURXT` contract together with its supporting `LibErrors` and `LibModifiers` libraries.

The codebase is concise and organizes its functionality into clearly delimited subsystems for issuance, redemption staging, blacklist management, regulatory seizure, pause control, and role administration, each gated by a dedicated access-control role. Inline NatSpec documentation and the supplementary client documents (covering the business specification, technical reference, governance policy, and operations manual) were sufficient to follow the intended end-to-end flow, including the off-chain redemption pipeline and the mapping between on-chain roles and the off-chain Taurus custody groups responsible for signing each operation. No issues of medium severity or higher were identified during the engagement. CACEIS has acknowledged the three low severity findings and has decided to incorporate the corresponding fixes into the next version of the smart contract, which will be the version taken to launch.

The audited material did not include an automated test suite, deployment scripts, or proxy initialization scripts. The supplied operations manual covers the deployment and post-deployment checklists in detail, including the proxy and `ProxyAdmin` configuration, the ordering of role grants, and the Taurus custody setup, so the absence of those scripts in the in-scope material is partially offset by the documented procedure. The internal test suite that CACEIS maintains around the contract was not included in scope, so the audit was unable to directly observe its coverage. The following areas are highlighted as warranting explicit coverage in the existing test suite ahead of deployment: exhaustive role-gating tests that confirm each privileged actor can only invoke the functions intended for that role and reverts on every other privileged entry point, branch coverage of every revert path in the contract and its libraries, full lifecycle coverage of the mint, transfer, blacklist, seize, burn, pause, and upgrade flows, and integration tests that exercise the trusted forwarder relay against each user-facing entry point. The supporting trust assumptions documented in the introduction, particularly those covering role management, the redemption-address lifecycle, and the off-chain settlement pipeline, should also be reflected in deployment runbooks and continuous monitoring so that any drift from the modeled behavior is detected early.

The OpenZeppelin team is grateful to CACEIS for their responsiveness, clear explanations, and thorough supporting documentation throughout the engagement.

Appendix

File-level Hashes

Initial Review

Below are the SHA-256 hashes for all individual files that were in scope and reviewed during the initial codebase review.

```
src/library/LibModifiers.sol
0db63bf03634352ef99a819a81da5b38974e325a09286bc4d24b78a6a8c7f7d7

src/library/LibErrors.sol
b26ee15d00bc2463ba566192fa0baefc7f3a5c286975eeaca5611dafa0f3e12b

src/EURXT.sol
8df99cfafbe614ceb18387448b5d5ca37d66059d7c421d22b0377073a4634319
```

Issue Classification

OpenZeppelin classifies smart contract vulnerabilities on a 5-level scale:

- Critical
- High
- Medium
- Low
- Note/Information

Critical Severity

This classification is applied when the issue's impact is catastrophic, threatening extensive damage to the client's reputation and/or causing severe financial loss to the client or users. The likelihood of exploitation can be high, warranting a swift response. Critical issues typically involve significant risks such as the permanent loss or locking of a large volume of users' sensitive assets or the failure of core system functionalities without viable mitigations. These issues demand immediate attention due to their potential to compromise system integrity or user trust significantly.

High Severity

These issues are characterized by the potential to substantially impact the client's reputation and/or result in considerable financial losses. The likelihood of exploitation is significant, warranting a swift response. Such issues might include temporary loss or locking of a significant number of users' sensitive assets or disruptions to critical system functionalities, albeit with potential, yet limited, mitigations available. The emphasis is on the significant but not always catastrophic effects on system operation or asset security, necessitating prompt and effective remediation.

Medium Severity

Issues classified as being of medium severity can lead to a noticeable negative impact on the client's reputation and/or moderate financial losses. Such issues, if left unattended, have a moderate likelihood of being exploited or may cause unwanted side effects in the system. These issues are typically confined to a smaller subset of users' sensitive assets or might involve deviations from the specified system design that, while not directly financial in nature, compromise system integrity or user experience. The focus here is on issues that pose a real but contained risk, warranting timely attention to prevent escalation.

Low Severity

Low-severity issues are those that have a low impact on the client's operations and/or reputation. These issues may represent minor risks or inefficiencies to the client's specific business model. They are identified as areas for improvement that, while not urgent, could enhance the security and quality of the codebase if addressed.

Notes & Additional Information Severity

This category is reserved for issues that, despite having a minimal impact, are still important to resolve. Addressing these issues contributes to the overall security posture and code quality improvement but does not require immediate action. It reflects a commitment to maintaining high standards and continuous improvement, even in areas that do not pose immediate risks.