OpenZeppelin | security

USDC Token Migration Audit

STARKWARE

November 28, 2025

Table of Contents

Summary

Type	DeFi
Timeline	From 2025-11-11 To 2025-11-12
Languages	Cairo

Total Issues	5 (4 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	0 (0 resolved)
Low Severity Issues	0 (0 resolved)
Notes & Additional Information	5 (4 resolved)

Scope

OpenZeppelin audited the starkware-libs/usdc-migration repository at commit de1489d.

In scope were the following files:

```
packages/token_migration/src/
├── errors.cairo
├── events.cairo
├── interface.cairo
├── lib.cairo
├── starkgate_interface.cairo
├── token_migration.cairo
└── utils.cairo
```

System Overview

The token migration contract has been designed to facilitate an upgrade of the USDC token that is widely used on Starknet. The USDC.e token is currently being used on Starknet, and it represents USDC that exists on Ethereum but has been locked and bridged to Starknet.

With the introduction of Circle's CCTP V2, USDC will no longer need to be bridged from Ethereum. Instead, Circle will implement minting and burning of USDC on all CCTP-V2-enabled chains. This will allow them to facilitate cross-chain transfers of assets, with no lock-up required. Within the context of these contracts, USDC.e is known as the "legacy" token, while the new CCTP-V2-enabled USDC is known as the "native" token. The native USDC token is meant to eventually replace the legacy token. To support this, the migration contract has been created.

This migration contract will allow users to send legacy USDC to it and receive native USDC in an equal amount. This contract has a switchable "reverse swap" capability, which allows users to swap from the native USDC back to the legacy one. The reverse swap can be enabled or disabled by the migration contract owner. The migration contract also maintains a "legacy buffer", which is a certain amount of legacy tokens that are kept in the migration contract. When this buffer is exceeded by at least one "batch size", a "batch" of legacy USDC is sent back across the token bridge to Ethereum, to a trusted L1 recipient contract.

Security Model and Trust Assumptions

The following trust assumptions were identified during the audit:

- It is assumed that the StarkGate bridge is operated correctly and that the migration contract will only be deployed with the correct addresses for legacy and native USDC tokens, as well as the correct L1 address being set for `l1_recipient`.
- It is assumed that the `owner` is a trusted entity controlled by Starkware, and the private keys associated with it are securely stored.

- It is assumed that all code outside the scope of this audit behaves as documented. This includes the StarkGate token bridge, the Staknet sequencer, and any contracts deployed on Ethereum.

Privileged Roles

The `owner` of the migration contract can control:

- upgrades to the migration contract
- all tokens that get transferred to the migration contract
- whether reverse swaps are enabled or disabled
- the size and number of batches for the legacy buffer, determining how many tokens remain on StarkNet, and when they are bridged to Ethereum
- the `l1_recipient` which receives the legacy tokens that are transferred back to Ethereum from StarkNet
- the `token_supplier` which provides either legacy or new tokens during migration

Notes & Additional Information

N-01 `allow_swap_to_legacy` Is Overloaded

The `allow_swap_to_legacy` identifier is both a storage variable and a function name. The function allows for the setting of the storage value with the same name.

To disambiguate the two and to follow the pattern of function names beginning with `set...`, consider renaming the `allow_swap_to_legacy` function to `set_allow_swap_to_legacy`.

Update: *Resolved in pull request #114 at commit 13bdbf0.*

N-02 Lack of Indexed Event Parameters

In `events.cairo`, multiple instances of events without indexed parameters were identified:

- The `L1RecipientVerified` event
- The `TokenSupplierSet` event
- The `LegacyBufferSet` event
- The `BatchSizeSet` event
- The `SendToL1Failed` event

Consider indexing event parameters using `#[key]` to improve the ability of off-chain services to search and filter for specific events.

Update: *Acknowledge, not resolved. The Starkware team stated:*

> *This is intentional. We don't expect any benefit from the events being indexed, and the presentation in the block explorer is worse, so we prefer them not being indexed.*

N-03 Missing Documentation

Throughout the codebase, multiple instances of missing documentation were identified:

- In `token_migration.cairo`, the `upgrade` function
- In `events.cairo`, the `TokenMigrated` event
- In `events.cairo`, the `L1RecipientVerified` event
- In `events.cairo`, the `TokenSupplierSet` event
- In `events.cairo`, the `LegacyBufferSet` event
- In `events.cairo`, the `BatchSizeSet` event
- In `events.cairo`, the `SendToL1Failed` event

Consider thoroughly documenting all functions and events, including their parameters and return values, that form part of a contract's public API.

Update: *Resolved in pull request #116 at commit c9970e7.*

N-04 Unnecessary Balance Checks

When executing the internal `swap` function, balance checks are performed for both the user executing the swap and the token supplier. However, these are unnecessary because, if the migrator tries to transfer more funds than held by the sender, the token implementation will make the transaction revert. Furthermore, since this migrator is designed exclusively for USDC.e and USDC, the behavior of the tokens is predictable and will revert unless the exact amount of tokens is successfully transferred. Hence, these are superfluous checks that only increase gas costs.

Consider removing the aforementioned balance checks. Alternatively, if the token migration contract is intended to ever be used with tokens that may charge fees or not revert upon failing transfers, consider keeping the checks in place and reflecting in the documentation the fact that the token migration contract is designed for general purposes (rather than just for USDC).

Update: *Resolved in pull request #115 at commit 36a5d2a. Documentation comments were added instead of changing or removing balance checks.*

> *The balance check are to stay. We added comments letting the reader better understand why.*

N-05 `utils.cairo` Is Empty and Unused

The `utils.cairo` file is empty and is unused within the codebase.

Consider removing `utils.cairo` to improve the clarity and maintainability of the code repository.

Update: *Resolved in pull request #113 at commit c73cf0e.*

Conclusion

The code under review comprises a token migration contract that is meant to aid Starknet users in migrating their USDC.e (legacy token) tokens to USDC (native token) without having to withdraw them on L1 themselves.

Overall, only a few issues were identified in this short engagement, with no critical- or high-severity issues reported. A few recommendations were also made to improve code quality and encourage thoughtful management of the contract once it goes live. The auditors were pleased to see a thoroughly documented and concisely designed codebase.

Beyond the recommendations made here, the Starkware team is encouraged to monitor the migration contract closely and implement plans to "shut down" the contract quickly if suspicious behavior is detected. Shutdown can be achieved by setting the `token_supplier` value to `0`. In addition, all transfers of tokens out of the migration contract should be monitored, and a corresponding, equal transfer of tokens into the migration contract should be ensured. Apart from this, it should be ensured that all batch transfers arrive on L1 correctly, and that the buffer is not exceeded by more than one batch size.

The Starknet team is appreciated for being highly responsive to the questions posed by the audit team and sharing comprehensive documentation about the project.

Appendix

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.