

Artist Token Contracts

Security Review

Cantina Managed review by:

M4rio.eth, Lead Security Researcher
MiloTruck, Lead Security Researcher

June 17, 2025

Contents

1 Introduction **2**
- 1.1 About Cantina . . . 2
- 1.2 Disclaimer . . . 2
- 1.3 Risk assessment . . . 2
  - 1.3.1 Severity Classification . . . 2

2 Security Review Summary **3**

3 Findings **4**
- 3.1 Medium Risk . . . 4
  - 3.1.1 Interest could remain locked when the underlying vault imposes withdrawal caps . . . 4
  - 3.1.2 Withdrawal-capped vaults can be DoSed via repeated migrations or flash-loans . . . 4
  - 3.1.3 Unsafe ERC-20 operations . . . 5
- 3.2 Low Risk . . . 5
  - 3.2.1 Hard-coded zero slippage exposes users to sandwich attacks . . . 5
  - 3.2.2 Withdrawal/deposit-fee vaults can break redemptions over time . . . 5
  - 3.2.3 `ArtistToken._migrate()` leaves dangling allowance to old vaults . . . 6
  - 3.2.4 Constant `MINT_PRICE` makes `ArtistToken` incompatible with vaults which may lose assets on deposit . . . 6
- 3.3 Informational . . . 7
  - 3.3.1 Deposits that mint zero shares silently break collateral accounting . . . 7
  - 3.3.2 Default `pragma` version should be `0.8.24` . . . 8
  - 3.3.3 Mixed access-control patterns create maintenance risks . . . 8
  - 3.3.4 Minor issues . . . 8
  - 3.3.5 Tiny migrations can exhaust the `ArtistToken`'s share backing via up-/down-rounding 9
  - 3.3.6 `ArtistToken._withdrawInterest()` does not consider deposits in old vaults . . . 9

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Royal's projects use crypto to create new forms of economic relationships between artists and their fans.

From Jun 1st to Jun 4th the Cantina team conducted a review of artist-token-contracts on commit hash ebb3640d. The team identified a total of **13** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	3	1	2
Low Risk	4	2	2
Gas Optimizations	0	0	0
Informational	6	4	2
Total	**13**	**7**	**6**

3 Findings

3.1 Medium Risk

3.1.1 Interest could remain locked when the underlying vault imposes withdrawal caps

Severity: Medium Risk

Context: ArtistToken.sol#L526-L534, ArtistToken.sol#L534

Finding Description: `ArtistToken.getWithdrawableInterest()` and the guard in `_withdrawInterest()` treat interest as.

```
maxWithdraw(vault) - principalDeposit   // floored at 0
```

`maxWithdraw()` returns only what the vault will let the contract pull out in the current block. If the underlying vault enforces a daily or epoch withdrawal cap, the value can remain below `principalDeposit` even when sizable yield has accrued. User principal stays safe, but the artist may be unable to claim yield for long periods, and analytics that read on-chain data will show interest as zero.

Recommendation: To solve this is not that easy because if we want to use the `previewRedeem` for example, we can block the principal deposit from being withdrawal if caps are applied. Or consider avoidinging such vaults altogether.

Royal.io: Acknowledged. For now, we plan to only allowlist `MetaMorpho` vaults, which do not have withdrawal caps. We are also adding "*no withdrawal caps*" to our internal vault approval criteria and have documented "*withdrawal-capped vaults*" as a variant of "*Incompatible Vaults*" in the `README`.

Cantina Managed: Acknowledged.

3.1.2 Withdrawal-capped vaults can be DoSed via repeated migrations or flash-loans

Severity: Medium Risk

Context: ArtistToken.sol#L553-L601

Finding Description: Currently there are two ways of DoS the withdrawals if the main vault is an ERC-4626 vault with an enforced daily/epoch withdrawal cap.

- The Flash-Loan Path: A malicious user can take a flash-loan, deposit the loan into the current vault, then immediately withdraw it.

  This round-trip consumes the entire withdrawal quota for the epoch while leaving the vault's share price unchanged. Repeating the flash-loan every epoch DoSes redemptions.

- The Migration Path: `ArtistToken._migrate()` lets the contract owner withdraw up to the vault's current `maxWithdraw()` and immediately deposit the same assets into a different approved vault.

  1. Drain the full quota from the current* vault (`oldVault`).
  2. Deposit into another vault (`newVault`).
  3. Switch the `vault` pointer to `newVault`.
  4. In the next block (or after the cap resets) perform the reverse migration, draining that quota too.

In both situations each vault's `maxWithdraw()` is kept at or near zero, so user-initiated burns will revert. Users must either wait for the cap to refresh and hope to front-run the DoSer to withdraw.

Recommendation: As this is the second issue related with withdrawal caps, consider avoiding such vaults.

Royal.io: Acknowledged. For now, we plan to only allowlist `MetaMorpho` vaults, which do not have withdrawal caps. We are also adding "*no withdrawal caps*" to our internal vault approval criteria and have documented "*withdrawal-capped vaults*" as a variant of "*Incompatible Vaults*" in the `README`.

Cantina Managed: Acknowledged.

3.1.3 Unsafe ERC-20 operations

Severity: Medium Risk

Context: ArtistToken.sol#L612, Withdrawable.sol#L94-L95

Description: The protocol calls ERC-20 functions without the use of `SafeERC20` in two places:

- In `ArtistToken._approveMaxVault()`:

```
bool success = IERC20(PAYMENT_TOKEN).approve(vaultToApprove, type(uint256).max);
if (!success) revert ApprovalFailed();
```

- In `Withdrawable._withdrawERC20()`:

```
bool success = erc20.transfer(receiver, amount);
if (!success) revert WithdrawalFailed();
```

As a result, these calls will revert for tokens that do not comply with the ERC-20 specification. An example would be USDT on Ethereum mainnet.

Recommendation: Consider using `forceApprove()` and `safeTransfer()` from `SafeERC20` respectively.

Royal.io: Fixed in commit 71479c97.

Cantina Managed: Fix verified.

3.2 Low Risk

3.2.1 Hard-coded zero slippage exposes users to sandwich attacks

Severity: Low Risk

Context: ArtistToken.sol#L234

Finding Description: Several functions hard-code a "no slippage" setting by passing 0 to the underlying safety parameters:

- `mint(address account, uint256 amount)` → `_depositAndMint(..., minSharesReceived = 0)`.
- `migrate(address newVault)` and `continueMigration(address oldVault)`.

Pass `minSharesReceived = 0`, `maxSharesRedeemed = type(uint256).max`.

When integrators or front-ends call these overloads, users get no protection against share-price movement. A miner/MEV bot can frontrun the transaction, deposit a large amount into the same vault, inflate the share price, and back-run their liquidity out, capturing the slippage difference (a classic sandwich / donation attack). Because the zero-slippage overload is the "easy" call, many developers will use it inadvertently, putting users at avoidable risk.

Recommendation: Delete the overloads that implicitly set slippage to 0 and keep only the full-parameter versions:

- `mint(account, amount, minSharesReceived)`.
- `migrate(newVault, maxSharesRedeemed, minSharesReceived, maxAmountToMigrate)`.
- `continueMigration(oldVault, maxSharesRedeemed, minSharesReceived, maxAmountToMigrate)`.

Anyone who truly wants zero protection can still pass 0 manually, but it will no longer be the silent default.

Royal.io: Fixed in commit 8405b529.

Cantina Managed: Fix verified.

3.2.2 Withdrawal/deposit-fee vaults can break redemptions over time

Severity: Low Risk

Context: ArtistToken.sol#L489-L498

Finding Description: `ArtistToken._burnAndWithdraw` subtracts `refundAmount` (assets) from `principalDeposit`, then calls:

```
sharesRedeemed = vault.withdraw(refundAmount, account, address(this));
```

If the underlying ERC-4626 vault charges a withdrawal fee, the call burns extra shares. No adjustment is made to `principalDeposit`, so share backing shrinks faster than the recorded principal. After enough user burns, `vault.balanceOf(this)` holds too few shares to redeem the remaining `principalDeposit`. The same behavior would occur during migration as well, because it also executes a `withdraw`, which would burn more shares due to the fees and would not cover the full `principalDeposit`. Furthermore, a vault that charges a deposit fee is even worse: the very first mint already gives the contract fewer shares than could cover the whole `principalDeposit`, which worsens the drift.

An example of this type of vault is Harmony One, though these aren't very common.

Recommendation: This is not an easy fix without changing a lot of the logic. The simplest solution would be to not accept these types of vaults.

Royal.io: Acknowledged. For now, we plan to only allowlist `MetaMorpho` vaults, which do not have withdrawal or deposit fees. We are also adding "*no withdrawal/deposit fee*" to our internal vault approval criteria and have documented "*withdrawal/deposit-fee vaults*" as a variant of "*Incompatible Vaults*" in the `README`.

Cantina Managed: Acknowledged.

3.2.3 `ArtistToken._migrate()` leaves dangling allowance to old vaults

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Context: ArtistToken.sol#L553-L601.

Description: When an `ArtistToken` is first deployed or when it is migrated to a new vault, `_approveMaxVault()` is called to grant infinite allowance to the vault:

```
function _approveMaxVault(address vaultToApprove) internal {
    if (IERC20(PAYMENT_TOKEN).allowance(address(this), vaultToApprove) == 0) {
        bool success = IERC20(PAYMENT_TOKEN).approve(vaultToApprove, type(uint256).max);
        if (!success) revert ApprovalFailed();
    }
}
```

This is required for making deposits into the vault. However, when migrating from an old vault to a new one in `_migrate()`, the infinite allowance to the old vault is not revoked. This could be dangerous if the allowance to the old vault is ever abused (e.g. old vault has an exploit).

Recommendation: In `_migrate()`, consider resetting the allowance to the old vault by calling `approve(oldVault, 0)`. However, note that this approach (i.e. calling `approve()` with zero amount) does not work for BNB on Ethereum mainnet, which reverts on zero approvals.

Royal.io: Fixed in commit b6887946.

Cantina Managed: Fix verified.

3.2.4 Constant `MINT_PRICE` makes `ArtistToken` incompatible with vaults which may lose assets on deposit

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Context: ArtistToken.sol#L438-L439.

Description: In `ArtistToken._depositAndMint()`, the amount of assets deposited into the vault is calculated from `MINT_PRICE` multiplied by the amount of tokens to mint:

```
uint256 totalPayment = MINT_PRICE * tokenAmount;
principalDeposit += totalPayment;
```

From the above, `totalPayment` is the amount of assets deposited while `tokenAmount` is the amount of `ArtistToken` minted. However, using a constant price of `MINT_PRICE` for deposits and withdrawals could be dangerous if a vault does not guarantee a user can withdraw the same amount of assets deposited. More specifically, for some vaults, it is possible for a user to lose assets when depositing under certain conditions. In such cases, it is the responsibility of the user to ensure they do not perform deposits which lose assets. An example would be a lending vault which is temporarily undercollateralized, and a portion of new deposits are used to socialize bad debt. If an `ArtistToken` uses such a vault, an attacker could drain the assets owned by the `ArtistToken` by repeatedly performing deposits and withdrawals. A naive example:

- Assume the following:
  - A lending vault has bad debt and 20% of new deposits are lost to socialize bad debt.
  - `ArtistToken` holds 1000 tokens as assets.
- Attacker deposits 100 tokens:
  - 80 tokens are owned by `ArtistToken` as shares.
  - 20 tokens are lost to the vault.
  - `ArtistToken` now holds 1080 tokens.
- Attacker withdraws 100 tokens:
  - `ArtistToken` now holds 980 tokens.

An attacker can repeatedly perform the above to drain all assets in the `ArtistToken` contract.

Recommendation: When whitelisting vaults in `ArtistTokenFactory`, ensure that users will always receive 100% of their assets as shares on deposit.

Royal.io: Acknowledged. For now, we plan to only allowlist `MetaMorpho` vaults, which do not have deposit fees (and do not socialize bad debt on deposit, which we view as a variant of a deposit fee). We are also adding "*no deposit fee*" to our internal vault approval criteria and have documented "*deposit-fee vaults*" as a variant of "*Incompatible Vaults*" in the `README`.

Cantina Managed: Acknowledged.

3.3 Informational

3.3.1 Deposits that mint zero shares silently break collateral accounting

Severity: Informational

Context: ArtistToken.sol#L449

Finding Description: `_depositAndMint` and `_migrate` accept a vault deposit even when `sharesReceived == 0`. This happens if the amount of PAYMENT_TOKEN sent is smaller than one share after `previewDeposit` rounding-down and the slippage is set to 0. Result:

- `principalDeposit` increases, but the contract holds no shares to cover that principal.
- A later burn or migration can revert or become under-collateralised.

The same risk applies to the deposit step inside `_migrate`: migrating a tiny or capped amount could mint zero shares in the new vault while still burning shares in the old one.

Recommendation: Add an explicit check after every vault deposit:

```
uint256 sharesReceived = vault.deposit(assets, address(this));
require(sharesReceived > 0, "zero shares received");
```

Do the same for the `sharesReceived` value returned by the deposit in `_migrate`. This guarantees each deposit always yields at least one share and prevents silent loss of collateral.

Royal.io: Fixed in commit 3d249ffc.

Cantina Managed: Fix verified.

3.3.2 Default `pragma` version should be `0.8.24`

Severity: Informational

Context: ArtistTokenFactory.sol#L2

Finding Description: `ArtistToken` (and related contracts) inherit `ReentrancyGuardTransient`, which relies on Solidity's transient-storage opcodes (`TSTORE`/`TLOAD`). These opcodes are only emitted by the compiler starting with **solc 0.8.24**, but the file header declares.

```
pragma solidity ^0.8.0;
```

This range allows compilation with older versions (e.g., 0.8.17) that do not generate transient opcodes, breaking the guard and causing compilation or runtime errors depending on the tool-chain.

Recommendation: Pin the `pragma` to a compiler that supports transient storage, for example:

```
pragma solidity ^0.8.24; // or a fixed 0.8.24/0.8.25
```

Royal.io: Fixed in commit 57c5c121.

Cantina Managed: Fix verified.

3.3.3 Mixed access-control patterns create maintenance risks

Severity: Informational

Context: ArtistTokenFactory.sol#L12

Finding Description: `ArtistTokenFactory` inherits `Ownable` and `AccessControl`:

```
contract ArtistTokenFactory is Ownable, AccessControl { /*...*/ }
```

- Owner-only functions: `addVault`, `removeVault`, `transferOwnership`.
- Role-based function: `createArtistToken` guarded by `DEPLOYER_ROLE`.
- `transferOwnership()` manually revokes and re-grants roles, but it does not grant the new owner `DEPLOYER_ROLE` (high-severity bug H-2 previously noted).

Maintaining two parallel permission systems increases complexity:

- Developers and auditors must reason about two sources of authority.
- A future refactor can forget to update one set of checks.
- Ownership transfer logic is fragile, as shown by the missing `DEPLOYER_ROLE` grant.
- Tooling (Timelock, Governor, multisig) must integrate with both patterns.

Recommendation: Adopt a single, consistent strategy:

- Drop `Ownable`.
- Use `DEFAULT_ADMIN_ROLE` as the "owner" authority or define a dedicated `OWNER_ROLE`.
- Replace `onlyOwner` modifiers with `onlyRole(OWNER_ROLE)`.

Royal.io: Fixed in commit 2ca83f67.

Cantina Managed: Fix verified.

3.3.4 Minor issues

Severity: Informational

Context: (*See each case below*)

Finding Description: We aggregated small issues in the code that do not require specific issues created.

- ArtistToken.sol#L693, ArtistToken.sol#L708: `getExpectedTotalInterest` and `getWithdrawableInterest` are declared `public`. Both functions are view-only and never called internally, so `external` would be better.

- ArtistTokenFactory.sol#L155: The factory computes the CREATE2 salt externally but does not include it in `ArtistTokenCreated`. Emitting the salt makes it easier for indexers and off-chain services to verify deterministic addresses and reproduce `create2` calculations.
- RefundExchange.sol#L240: The `depositFunds` in the `RefundExchange` could be removed as it does not add any functionality, the token could be transferred outside to the `RefundExchange`.

Recommendation: Consider fixing the aforementioned issues.

Royal.io: Fixed in commit 8cbf388d.

Cantina Managed: Fix verified.

3.3.5 Tiny migrations can exhaust the `ArtistToken`'s share backing via up-/down-rounding

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Finding Description: Each call to `_migrate` burns shares rounded up in `oldVault.withdraw` and mints shares rounded down in `newVault.deposit`. For very small `migrateAmount` (dust-level), the mismatch is at least `1 wei` of shares and can become problematic if the artist decided to weaponize it.

```
function _migrate(
    address oldVault,
    address newVault,
    uint256 maxSharesRedeemed,
    uint256 minSharesReceived,
    uint256 maxAmountToMigrate
) internal {
    // ...

    // Withdraw from old ERC4626 vault
    uint256 sharesRedeemed = IERC4626(oldVault).withdraw(migrateAmount, address(this), address(this)); //
    ↪  rounds up
    if (sharesRedeemed > maxSharesRedeemed) revert ExcessiveSharesRedeemed();

    // Deposit into new ERC4626 vault
    _approveMaxVault(newVault);
    uint256 sharesReceived = IERC4626(newVault).deposit(migrateAmount, address(this)); // rounds down
    if (sharesReceived < minSharesReceived) revert InsufficientSharesReceived();
    // ...
}
```

An artist who repeatedly performs such dust migrations can linearly drain the ArtistToken's share balance until it no longer covers the remaining `principalDeposit`. At that point user burns and future migrations will revert. This won't result in permanent losses because anyone can transfer additional vault shares to the ArtistToken contract to "re-collateralise it". However this relies on off-chain monitoring and manual action and additional purchase of shares to cover the deficit; until the donation is made, users remain blocked.

Recommendation: As this is hard to combat because every solution has pros and cons; for example, adding a minimum amount to migration could leave shares locked in the contract if they fall below that threshold.

Royal.io: Acknowledged. This issue would require many iterations for losses to accumulate to at least 1 USDC, making it impractical to exploit. Additionally, only the artist can perform this griefing attack, and there is no incentive for them to do so.

Cantina Managed: Acknowledged.

3.3.6 `ArtistToken._withdrawInterest()` does not consider deposits in old vaults

Severity: Informational

Context: ArtistToken.sol#L533-L534, ArtistToken.sol#L666-L668.

Description: In `ArtistToken._withdrawInterest()`, the following check ensures the artist can only withdraw assets outside of `principalDeposit` by checking that `getWithdrawalBalance()` is not less than `principalDeposit` at the end of the function:

```
// The artist is not able to dip into user deposits when withdrawing interest
if (getWithdrawableBalance() < principalDeposit) revert InsufficientPrincipal();
```

However, `getWithdrawableBalance()` only returns the amount of assets in the current `vault`:

```
function getWithdrawableBalance() public view returns (uint256 vaultAssets) {
    return getWithdrawableBalance(address(vault));
}
```

As such, if a partial migration occurs, assets remaining in the old vault will not be included in `getWithdrawableBalance()`. This causes the check in `_withdrawInterest()` to underestimate the amount of assets owned by the contract, hence the artist can only withdraw less interest than expected.

For example:

- Old vault has 100 tokens.
- New vault has 100 tokens.
- `principalDeposit` is equal to 100 tokens.
- Even though there are 100 excess tokens, the artist cannot withdraw any interest.

Recommendation: Consider modifying `getWithdrawableBalance()` to include assets in all previously used vaults.

Royal.io: This behavior is intentional. We want to incentivize artists to perform full migrations between vaults whenever possible rather than intentionally splitting assets across multiple vaults.

Cantina Managed: Acknowledged.