Odx Contracts

Security Review

Cantina Managed review by:

R0bert, Lead Security Researcher
Blockdev, Security Researcher

May 12, 2025

Contents

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

Odx is the first permissionless perpetual protocol with custom zkEVM App-chain.

From Apr 25th to Apr 29th the Cantina team conducted a review of odx-contracts on commit hash e3f2850b. The team identified a total of **13** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	2	2	0
Medium Risk	0	0	0
Low Risk	4	2	2
Gas Optimizations	1	1	0
Informational	6	4	2
Total	**13**	**9**	**4**

3 Findings

3.1 High Risk

3.1.1 `_mint()` can mint more than the limit

Severity: High Risk

Context: Controller.sol#L303-L316

Description: `_mint()` first calculates the USD value of the `amount` tokens to mint using `_calculateMintValue()`. If the maximum remaining limit of the epoch is lower than the USD value of `amount` tokens, then it returns a lower value. `_mint()` function then updates the limit of the minter and witnesses by this value, but still mints the full amount to the `destination`. Thus, in the worst case, even if `_calculateMintValue()` returns 0, the full amount is minted.

Recommendation: Either revert if minting the entire `amount` exceeds the limit or mint a lower amount.

Odx: Fixed in commit 95fddb8.

Cantina Managed: Fix verified.

3.1.2 Storage variables are set in constructor instead of in `initializer()`

Severity: High Risk

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

Description: `Controller` and `Oracle` contract are upgradeable contracts. Upgradeable contracts should initialize their storage variables only in `initializer()` since any initialization of these variables in constructor doesn't affect the proxy storage. However, these contracts call `AccessControlDefaultAdminRules()` and `Ownable()` constructor setting key variables.

Recommendation:

- Use the upgradeable version of all the OZ contracts from openzeppelin-contracts-upgradeable.
- Add `_disableInitializers()` (imported from `Initializable` contract) to the constructors to block anyone to call `initializer()` on these implementation contracts.

Odx: Fixed in commit e6b7594.

Cantina Managed: Fix verified.

3.2 Low Risk

3.2.1 No partial fill in mint requests

Severity: Low Risk

Context: Controller.sol#L318

Description: The `_calculateMintValue` function forces the merchant and each witness to have exactly the same remaining "USD mint budget" for the requested amount whenever one of them surpasses the mint limit. If any participant has a smaller budget, the function reverts and the entire mint operation fails. This design means there is no mechanism for partial fulfilment of a mint request if one signer has a lower limit.

For example:

- Minting amount: 1000 USD.
- Merchant remaining mint limit for current epoch: 900.
- Witness #1 mint limit for current epoch: 900.
- Witness #2 mint limit for current epoch: 950.

The `request` will revert with a `MintLimitReached` error instead of allowing the mint of 900 USD worth in tokens.

Recommendation: If the goal is to prevent any partial minting and ensure full alignment across all signers, no code change is needed. However, if partial mints are a desired feature (for example, allowing the merchant to mint only what each witness can also mint without surpassing their limit), consider modifying `_calculateMintValue` to handle partial amounts. This might involve returning and minting only the smallest leftover budget among the participants (the merchant and all the witnesses) rather than reverting entirely.

Odx: Acknowledged. We want to prevent any partial minting, since the system around this is not setup to handle partial mints.

Cantina Managed: Acknowledged.

3.2.2 Request calls can be front-run by another merchant

Severity: Low Risk

Context: Controller.sol#L131

Description: The `request(Request memory req, bytes[] memory witnessSigs)` function checks `hasRole(MERCHANT_ROLE, msg.sender)` but does not enforce `req.account == msg.sender`. Consequently, a merchant (`MerchantA`) can call request using `req.account = MerchantB` (as long as `MerchantB` also has the `MERCHANT_ROLE`). Although this does not allow `MerchantA` to mint funds in `MerchantB`'s name (because the minted tokens still go to `req.account` specified in the request), it can be used to "front-run" or otherwise interfere with `MerchantB`'s intended batch operation if `MerchantB` was about to call the same request.

A potential problem with this implementation is that if the front-running merchant was able to have access somehow to the witness signatures given to the legit merchant, the front-running merchant could submit the request call in an epoch where both (the legit merchant and all the witnesses) are over the mint limit. This way the signatures would be invalidated but the legit merchant would not receive any minted tokens (it would be minted 0 tokens). Whenever `witnessVal` or `mintVal` is equal to 0 the transaction should revert to avoid this scenario from happening.

Recommendation: If the design requires that each merchant only submit requests for their own account, consider adding `require(req.account == msg.sender)` within the `request` function implementation. That ensures no one can front-run or submit a request "on behalf" of another merchant. On the other hand, consider reverting whenever `witnessVal` or `mintVal` is equal to 0.

Odx: Fixed in commit e3c9f4e.

Cantina Managed: Fix verified.

3.2.3 Controller excludes smart contract wallets signatures

Severity: Low Risk

Context: Controller.sol#L387

Description: Within the `Controller._validateRequest` function, userd for signature verification, the code snippet.

```
function _validateRequest(Request memory req, bytes[] memory witnessSig) internal returns (address[] memory) {
    uint256 operationsLength = req.operations.length;
    uint256 witnessLength = witnessSig.length;
    bytes32 digest = hashRequest(req);
    uint256 nonce = req.nonce;
    address merchant = req.account;
    address[] memory witnesses = new address[](witnessLength);

    if (req.expiration < block.timestamp) revert SignatureExpired();
    if (req.expiration > block.timestamp + EPOCH_DURATION) revert SignatureTooFarInFuture();
    if (witnessLength != witnessThreshold) revert InvalidWitnessCount();
    if (operationsLength == 0) revert NoOperations();

    _checkRoleAndPaused(merchant, MERCHANT_ROLE);
    _useUnorderedNonce(merchant, nonce);

    for (uint256 i = 0; i < witnessSig.length; ++i) {
        address witness = digest.recover(witnessSig[i]); // <<<
        _checkRoleAndPaused(witness, WITNESS_ROLE);
        _useUnorderedNonce(witness, nonce);
        witnesses[i] = witness;
    }

    for (uint256 i = 0; i < operationsLength; ++i) {
        Operation memory op = req.operations[i];
        _validateOperation(op);
    }
    return witnesses;
}
```

relies on `ECDSA.recover` which only supports externally owned accounts (EOAs). As a result, contract wallets (including multisigs) cannot generate valid signatures under this scheme. This restriction excludes a significant user base that employs smart contract wallets for automation, treasury management and advanced DeFi interactions. Any operator behind a multisig will not be able to provide a valid signature and operate as a witness.

Recommendation: Use a more flexible approach that accommodates both EOAs and contract wallets. For instance, OpenZeppelin's `SignatureChecker.isValidSignatureNow` can verify signatures from EOAs while also supporting ERC1271 for contract accounts. By switching to a signature verification method compatible with contract wallets, you ensure that operators behind a multisig could provide their signature to operate as a witness.

Odx: Acknowledged. At this point in time, we are not looking to support smart-contract wallet signatures.

Cantina Managed: Acknowledged.

3.2.4 Use `__gap` to reserve slots for future storage variables

Severity: Low Risk

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

Description: Since ODX contracts are supposed to be used as proxy implementations, introducing new storage variables may misalign the existing storage variables with the storage slots they currently occupy.

Recommendation:

1. Add `uint[50] private __gap` to each contract.
2. If introducing a new storage variable, declare it just before `__gap` declaration, and decrement the size of `_gap` by 1.

Odx: Fixed in commit 066ff4d and 16aab29.

Cantina Managed: Fix verified.

3.3 Gas Optimization

3.3.1 Simplify nested `if`/`else` blocks

Severity: Gas Optimization

Context: Controller.sol#L338-L346

Description: By rearranging, the `if` conditions in the highlighted code, the nested code can be simplified.

Recommendation: Refactor the code as follows:

```
if (value + epoch.minted <= limitVal) {
    return value;
}
if (epoch.minted < limitVal) {
    return limitVal - epoch.minted;
} else {
    return 0;
}
```

Odx: Fixed in commit 129bb7d.

Cantina Managed: Fix verified.

3.4 Informational

3.4.1 Lack of staleness checks for oracle data

Severity: Informational

Context: Oracle.sol#L53

Description: Within `ODXOracle`, the contract calls `priceFeed.latestRoundData()` to obtain the Chainlink aggregator's latest price. However, it does not verify how recently or frequently the price feed has been updated. In other words, the contract does not check for staleness (e.g., by comparing the aggregator's updatedAt time to the current block timestamp). If an oracle is not updating properly for an extended period, the system may continue relying on outdated price data.

While it may be acceptable for an approximate USD conversion in this system, stale price data could cause inaccurate calculations of value. If the feed is stuck at an old price, merchants or witnesses might either overestimate or underestimate the actual USD value tied to tokens, skewing the system's mint limit checks.

Recommendation: If a high degree of price accuracy is crucial, consider including a staleness check by inspecting the aggregator's latest timestamp (e.g., `updatedAt`). Revert or take other precautions if the data is older than a configured threshold. If minor deviations are acceptable, you can keep this as is but add external monitoring to ensure the feed remains active.

Odx: Acknowledged. As already stated, we are using this oracle just to get an approximate USD value. Nothing precise, hence choosing to consider this issue for the next version.

Cantina Managed: Acknowledged.

3.4.2 Unneeded sequencer uptime check

Severity: Informational

Context: Oracle.sol#L50

Description: Within the `ODXOracle`, the contract calls `_checkSequencer` prior to fetching price data from the Chainlink aggregator. `_checkSequencer` verifies the L2 sequencer is active and that the configured grace period has passed. This step is commonly required on optimistic rollups (e.g., Optimism) to ensure that data feeds are reliable after the sequencer recovers from downtime. However, if this contract is deployed on a chain that does not rely on such a sequencer, this check may be an unnecessary overhead.

Recommendation: If the design specifically requires L2 downtime checks (e.g., for an Optimistic Rollup environment), keep `_checkSequencer` as is. Otherwise, consider making the sequencer check optional or configurable so it can be disabled on networks where no sequencer-based downtime risk exists.

Odx: Fixed in commit 3651f23.

Cantina Managed: Fix verified.

3.4.3 Missing event emission in setter functions

Severity: Informational

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

Description: Several functions in the codebase modify critical state variables without emitting any corresponding event. Specifically:

- `src/Controller.sol:277`: The line `witnessThreshold = threshold;` changes the threshold of witness signatures required but does not emit an event to notify off-chain systems.
- `src/Oracle.sol:92`: The line `feeds[asset] = Feed(feedType, feedData);` updates or sets the feed configuration for a given asset but does not emit an event to track these changes.
- `src/WrappedAsset.sol:86`: The function `setUserBlacklist(address user, bool blacklist)` modifies the blacklist status of a user but does not emit an event reflecting that change.

Without events, external monitors, user interfaces, or analytics tools have to rely on raw transaction logs or chain state diffs to detect these updates. Emitting events at critical points is a best practice for transparency, debugging and integration with off-chain services.

Recommendation: Add dedicated event declarations for each state-modifying function above and emit them whenever changes occur.

Odx: Fixed in commit 24ebd41.

Cantina Managed: Fix verified.

3.4.4 Unused code

Severity: Informational

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

Description: Certain parts of the codebase declare imports or custom errors that are never referenced within the contracts, becoming "dead code." Specifically:

- `src/Controller.sol:12` imports `IERC20` from "openzeppelin-contracts/token/ERC20/IERC20.sol" but never actually uses it in the contract.
- `src/Oracle.sol:12` declares the custom error `InvalidFeedType()` which is never thrown or referenced within the code.

Leaving unused imports or custom errors in the code does not typically pose a security risk. However, it can lead to confusion, clutter the codebase, and require unnecessary attention during future maintenance or audits.

Recommendation: Remove the unused `IERC20` import and `InvalidFeedType` error.

Odx: Fixed in commit 470bf73.

Cantina Managed: Fix verified.

3.4.5 Use `Math.min()`

Severity: Informational

Context: Controller.sol#L329-L334

Description: The highlighted code returns the minimum of `value` and `limitVal`. Since OZ Math is already imported, `Math.min()` can be used instead.

Recommendation: Return the min value using:

```
return Math.min(value, limitVal);
```

Odx: Fixed in commit 5325995.

Cantina Managed: Fix verified.

3.4.6 Add tests

Severity: Informational

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

Description: Unit tests are important to catch bugs and also future proof the code from the same bugs. We highly recommend spending time on adding unit and fuzz tests testing for usual and unusual behavior.

Recommendation: Add tests aiming for high coverage.

Odx: Acknowledged. We have currently an internal team writing tests for this. It's not in this same repository. Tests will be formalized and added to this repository in the next 14-30 days.

Cantina Managed: Acknowledged.