Glow Offchain Fractions

Security Review

Cantina Managed review by:

Optimum, Lead Security Researcher
R0bert, Lead Security Researcher

September 25, 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 `buyFractions()` has limited support for tokens like USDG and GLOW . . . 4
  - 3.1.2 Refunds do not fully support GLOW and USDG tokens . . . 4
- 3.2 Low Risk . . . 5
  - 3.2.1 Zero-effective buys allowed when sold out emit misleading events . . . 5
  - 3.2.2 `setRefundDetails` should reject updates once sale is non-refundable . . . 5
- 3.3 Gas Optimization . . . 5
  - 3.3.1 `FractionData.owner` is not used in the contract . . . 5
  - 3.3.2 Transient storage can be used for the `ReentrancyGuard` . . . 6
- 3.4 Informational . . . 6
  - 3.4.1 `buyFractions()` is missing the check of `minStepsToBuy <= stepsToBuy` . . . 6
  - 3.4.2 `_validateFractionCreationParams()` missing input validation checks . . . 6
  - 3.4.3 `claimRefund()`: Consider allowing a privileged user to claim the refund on behalf of the buyer . . . 6
  - 3.4.4 `closeFraction()` is callable for expired fraction sales . . . 7
  - 3.4.5 Post-threshold transfers bypass the no-tax guard . . . 7
  - 3.4.6 `closeFraction` authorization doc/code mismatch . . . 8
  - 3.4.7 Unused custom errors . . . 8
  - 3.4.8 Misleading custom error names . . . 8
  - 3.4.9 Unnecessary CFH resolution on non-payout paths . . . 9
  - 3.4.10 `buyFractions` ignores `creditTo` for refunds/ownership lookup . . . 9
  - 3.4.11 Shared CFH escrow queue across multiple sales on same token . . . 9
  - 3.4.12 Clarify pre-threshold escrow destination when CFH is enabled . . . 10
  - 3.4.13 Remove lingering `@auditor` comment . . . 10
  - 3.4.14 Limit sale expiration to avoid indefinite capital lock-up . . . 10

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 level	Impact: High	Impact: Medium	Impact: Low
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

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 are 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

GlowSwap is a decentralized exchange (DEX) protocol built on a two token pair constant-product automated market maker (CPAMM) foundation. GlowSwap allows anyone to borrow the underlying liquidity in a pair in return for interest payments on the amount of liquidity they borrowed.

From Sep 11th to Sep 12th the Cantina team conducted a review of glow-contracts on commit hash bee62881. The team identified a total of **20** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	2	2	0
Low Risk	2	2	0
Gas Optimizations	2	1	1
Informational	14	11	3
Total	**20**	**16**	**4**

3 Findings

3.1 Medium Risk

3.1.1 `buyFractions()` has limited support for tokens like USDG and GLOW

Severity: Medium Risk

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

Description: `OffchainFractions` is expected to support non tax/rebase tokens as tokens used to facilitate the sale of fractions defined as `FractionData.token`. This category is supposed to include tokens like USDG and GLOW as well. The issue is that these two tokens (issued by the Glow team) can only be transferred to/from selected set of contracts, and neither `OffchainFractions` is part of this set, nor any contract caller `buyFractions()` that might be a contract. To put simply, `_handlePurchaseTransfers()` will revert on trying to pull these tokens from `msg.sender` to `address(this)` since the both the caller (in case of a contract) and the `OffchainFractions` contract itself are not whitelisted in the USDG and GLOW contracts, as we can see in the following snippet:

```
// _handlePurchaseTransfers()
// ...
if (details.newFractionsSold < minSharesToRaise) {
    // Below minimum threshold - hold funds in contract
    _safeTransferFromNoTaxToken(token, msg.sender, address(this), details.amount); // audit-note: will revert
}
// If `minShares` has been reached
// If it's the first time reaching `minShares`, transfer all accumulated funds to the recipient. and mark it as
↪  claimed.
// If it's not the first time reaching `minShares`, transfer the funds to the recipient.
else {
    // Above minimum threshold - handle fund distribution
    if (fraction.claimedFromMinSharesToRaise) {
        // Minimum already claimed, send directly to recipient
        IERC20(token).safeTransferFrom(msg.sender, details.sendTo, details.amount);
    } else {
        // First time reaching minimum - transfer all accumulated funds
        _safeTransferFromNoTaxToken(token, msg.sender, address(this), details.amount); // audit-note: will
        ↪  revert
        // ...
    }
}
```

It is important to mention that this issue will not cause loss of funds, but rather the failure of calls to the contract for these two tokens. There is also a work-around to allow the support of these tokens (only for EOA buyers) only in case the seller declares the fraction sale with `minSharesToRaise=0`.

Recommendation: Consider using `getCurrentCFH()` to tackle this issue, the same way it is used inside `_calculatePurchaseDetails()` for the other code flows as well, make sure it is also applied in case of non-EOA buyers (callers of `buyFractions()`). As part of the fix, `_finalizePurchase()` should also be fixed to support emitting the `FractionSold` event with a flexible `buyer` instead of the current code that always uses `msg.sender` as the `buyer`:

```
emit FractionSold(id, creator, msg.sender, fraction.step, details.amount);
```

The reason for allowing a flexible buyer is because in order to allow contracts to call `buyFractions()`, their corresponding `CounterfactualHolder` contract should be the one calling `buyFractions()` on behalf of them, and since it is a one-time address, future funds that will be later sent to this address will not be claimable and therefore, we need to have a flexible recipient address.

Glow: Fixed in commit 92d5166.

Cantina Managed: Fix verified. `OffchainFractions` is now supporting both tokens fully, `_handlePurchaseTransfers()` allows receiving both tokens in all code flows, `FractionSold` is emitted with the address to credit.

3.1.2 Refunds do not fully support GLOW and USDG tokens

Severity: Medium Risk

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

Description: Similarly to "`buyFractions()` has limited support for tokens like USDG and GLOW", the refund mechanism does not fully support GLOW and USDG tokens. The issues are as follows:

1. The current implementation of `claimRefund()` only allows refunds to be claimed from the original externally owned account (EOA) that called `buyFractions()`.

```
function claimRefund(address creator, bytes32 id) external nonReentrant {
    // audit-note: only buyers can call this function
    uint256 _stepsPurchased = stepsPurchased[msg.sender][creator][id];
    // ...
    // audit-note: credits only the original caller of `buyFractions()`
    IERC20(fraction.token).safeTransfer(msg.sender, amount);
    emit FractionRefunded(id, creator, msg.sender, amount);
}
```

This restriction causes refunds to become inaccessible for contract-based buyers, since those have to be `CounterfactualHolder` contracts that cannot later invoke `claimRefund()` because they are one-time use only.

2. Incompatibility with USDG/GLOW transfers. At the end of `claimRefund()`, transferring USDG or GLOW tokens will fail for two reasons:
3. `OffchainFractions` is a non-whitelisted contract, so transfers of these tokens from it revert.
4. If the refund recipient is a contract, the transfer will also fail.

```
// claimRefund()
// ...
IERC20(fraction.token).safeTransfer(msg.sender, amount);
// ...
```

Recommendation:

1. Extend `claimRefund()` to support delegated refund claims, where the refund can be claimed by a designated address defined by the original CounterfactualHolder (CFH) contract used in `buyFractions()`.
2. Modify `claimRefund()` to source refunds from the CFH contract's balance rather than `OffchainFractions`, and ensure that transfers to CFH contracts are supported when the refund recipient is itself a contract.

Glow: Fixed in commit b92439b.

Cantina Managed: Fix verified. Refunds now support GLOW and USDG tokens.

3.2 Low Risk

3.2.1 Zero-effective buys allowed when sold out emit misleading events

Severity: Low Risk

Context: OffchainFractions.sol#L360-L367

Description: When the round is sold out (`stepsLeft == 0`), a buyer can still call `buyFractions` with `minStepsToBuy == 0` and pass the guard. The code then sets `details.stepsToBuy = 0` and proceeds to finalize, emitting `FractionSold` with `amount == 0` and re-emitting `RoundFilled`, even though no state changes occur.

```
// ...
    if (stepsLeft < minStepsToBuy) revert InsufficientSharesAvailable();
    details.stepsToBuy = min(stepsLeft, stepsToBuy);
}

details.newFractionsSold = fraction.soldSteps + details.stepsToBuy;
details.amount = details.stepsToBuy * fraction.step;

details.roundFullyFilled = details.newFractionsSold == fraction.totalSteps;
```

This can mislead indexers and monitoring and degrade UX: callers pay gas for what looks like a successful purchase, but it's a no-op (zero amount/steps credited), which is unexpected.

Recommendation: After computing `details.stepsToBuy`, revert if it is zero to block zero-effective buys. Alternatively, add an explicit sold-out guard (`if (stepsLeft == 0) revert InsufficientSharesAvailable();`) to prevent any further buys once the round is full.

Glow: Fixed in commit 2be3194.

Cantina Managed: Fix verified.

3.2.2 `setRefundDetails` should reject updates once sale is non-refundable

Severity: Low Risk

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

Description: `setRefundDetails` lets users override their refund destination even after a fraction sale has already cleared the minimum raise, which means `claimRefund` will always revert. While this doesn't break fund safety, the refund path itself refuses to run once `soldSteps >= minSharesToRaise`, it can mislead users or integrators who might assume a late `setRefundDetails` call will take effect. Aligning the behaviour with `buyFractions` would keep the functionality consistent and make a no-op state explicit.

Recommendation: Add a guard to `setRefundDetails` that reverts if the sale has already reached its minimum or has been marked as non-refundable, so callers receive a clear signal rather than silently setting unusable refund metadata.

Glow: Fixed in commit ff44378.

Cantina Managed: Fix verified.

3.3 Gas Optimization

3.3.1 `FractionData.owner` is not used in the contract

Severity: Gas Optimization

Context: OffchainFractions.sol#L175

Description: In the current version of the code, `FractionData.owner` is defined as the caller of `createFraction()` and can't be changed during the lifecycle of this specific fraction. `getFraction()` requires the caller to specify the creator which is always the original owner.

Recommendation: Consider removing the `owner` field as long as changing ownership for a fraction is not a feature that the team is willing to support.

Glow: Fixed in commit dac6b9e.

Cantina Managed: Fix verified.

3.3.2 Transient storage can be used for the `ReentrancyGuard`

Severity: Gas Optimization

Context: OffchainFractions.sol#L7

Description: The library ReentrancyGuard is used. There is also a version that uses transient storage that is more efficient.

Recommendation: Consider using ReentrancyGuardTransient Ensure the chain on which the contract are deployed support transient storage.

Glow: Acknowledged. We are on an old version of OZ and currently we do not want to upgrade to a new version.

Cantina Managed: Acknowledged.

3.4 Informational

3.4.1 `buyFractions()` is missing the check of `minStepsToBuy <= stepsToBuy`

Severity: Informational

Context: OffchainFractions.sol#L204

Description: `buyFractions()` allows callers to specify how many steps to buy (`stepsToBuy`) while they can also specify the minimum expected amount `minStepsToBuy`, the current version of the code is missing a check that verifies `minStepsToBuy <= stepsToBuy`.

Recommendation: Consider adding the recommended check.

Glow: Fixed in commit c7ed5dc.

Cantina Managed: Fix verified.

3.4.2 `_validateFractionCreationParams()` missing input validation checks

Severity: Informational

Context: OffchainFractions.sol#L311

Description: `_validateFractionCreationParams()` is missing the following input validation checks:

1. `expiration >= block.timestamp`.
2. `closer != address(0)`.

Recommendation: Consider implementing the proposed checks.

Glow: Fixed in commit 55f8a7e.

Cantina Managed: Fix verified.

3.4.3 `claimRefund()`: Consider allowing a privileged user to claim the refund on behalf of the buyer

Severity: Informational

Context: OffchainFractions.sol#L230

Description: `claimRefund()` allows buyers that already transferred the tokens for buying fractions to claim them back in case the round was not filled and either the fraction was declared closed by its owner (`closer`) or is expired. Either way, users will have to track the state of the fraction sale and that might add friction to the process which might result in unclaimed refunds.

Recommendation: Consider introducing a permissioned role, controlled by the Glow team, that can claim refunds on behalf of users. This role should only be able to execute the refund to the exact address recorded in the `stepsPurchased` mapping for that user. This ensures refunds can only ever be claimed by the buyer themselves or by the authorized Glow role acting strictly as a proxy.

Glow: Fixed in commit 92d5166.

Cantina Managed: Fix verified: `claimRefund()` now allows users to delegate refund claims to a chosen operator or any arbitrary address on their behalf.

3.4.4 `closeFraction()` is callable for expired fraction sales

Severity: Informational

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

Description: `closeFraction()` allows fraction sales creators (`FractionData.closer`) to close fraction sales which means to disable future buy orders (calls to `buyFractions()` for that specific fractions sale) as well as enable buyers to claim refunds by calling `claimRefund()`. The scenario of calling `closeFraction()` for expired fraction sales is therefore redundant since either-way the result will be the same, i.e. `buyFractions()` will be disabled and `claimRefund()` will be enabled.

Recommendation: Consider restricting `closeFraction()` so it can only be called before the fraction sale's expiration. While allowing closure after expiration has no functional impact, enforcing this check helps

avoid unnecessary or unintended state changes and aligns the function's behavior with expected lifecycle boundaries.

Glow: Fixed in commit 39450f4.

Cantina Managed: Fix verified.

3.4.5 Post-threshold transfers bypass the no□tax guard

Severity: Informational

Context: OffchainFractions.sol#L388-L410

Description: After the minimum threshold is reached, or when `minSharesToRaise == 0` (which sets `claimedFromMinSharesToRaise = true` at creation), the buy path routes funds without enforcing "no-tax" semantics or verifying what the recipient actually receives. In the recurring post-threshold path, funds go directly from the buyer to the recipient via `safeTransferFrom`. In the "first time reaching minSharesToRaise" path, funds are pulled into the contract, then forwarded to the recipient with a plain `safeTransfer`. Neither path checks the recipient's net delta.

With fee-on-transfer tokens, the recipient will receive less than `details.amount` or `totalAmount` while the contract still credits full "step" amounts, breaking economic/accounting assumptions. Note that some tokens can toggle fees on/off at any time. Even if a token is non-tax when the round starts, it may later enable fees, so the only robust defense is to verify net amounts on every transfer.

Recommendation: Merely informative. There is no reliable on-chain way to detect whether an ERC□20 can enable or toggle transfer fees in the future. Document this limitation clearly and communicate that tokens with dynamic or fee-on-transfer behavior are unsupported and may cause underpayment if fees are switched on later.

Glow: Acknowledged. The no tax is only to ensure that if refunds are necessary that we can process them without breaking the state machine. There will probably never be a tax token accepted in the `OffchainFractions` contract.

Cantina Managed: Acknowledged.

3.4.6 `closeFraction` authorization doc/code mismatch

Severity: Informational

Context: OffchainFractions.sol#L267

Description: The NatSpec above `closeFraction` claims "Only the creator can close their own fraction sale," but the function actually authorizes the configured closer address. The check enforces `msg.sender == fraction.closer`, not `msg.sender == creator` or `fraction.owner`:

```
/**
- @dev Only the creator can close their own fraction sale
*/
function closeFraction(address creator, bytes32 id) external nonReentrant {
    FractionData storage fraction = _fractions[creator][id];
    if (msg.sender != fraction.closer) {
        revert NotFractionsCloser();
    }
    // ...
}
```

This discrepancy can mislead integrators and operators into assuming creator-only permissions when a different closer was set at creation, causing failed ops or misconfiguration.

Recommendation: Consider updating the NatSpec comment to be aligned with the current implementation.

Glow: Fixed in commit 2be3194.

Cantina Managed: Fix verified.

3.4.7 Unused custom errors

Severity: Informational

Context: OffchainFractions.sol#L18, OffchainFractions.sol#L41, OffchainFractions.sol#L46-L48

Description: The contract declares several error types that are never used anywhere in the current implementation:

```
error NotFractionsOwner();
error CannotClaimPayoutWhenRoundNotFullyFilled();
error AlreadyClaimed();
error NotAllOrNothing();
error AlreadySent();
```

These errors are not referenced in any require/revert paths. Keeping dead error types adds deployment cost and can mislead integrators about supported states or features (e.g., "claim payout" or "all-or-nothing" flows that aren't implemented).

Recommendation: Consider removing the unused custom errors listed above.

Glow: Fixed in commit f90a0f8.

Cantina Managed: Fix verified.

3.4.8 Misleading custom error names

Severity: Informational

Context: OffchainFractions.sol#L241-L243, OffchainFractions.sol#L282-L284

Description: Two error names use "full/fully filled" but are thrown when only the soft cap (`minSharesToRaise`) is reached, not when the round is completely sold out (`totalSteps`). In `claimRefund` the guard is based on reaching the threshold, yet the error reads `CannotClaimRefundWhenRoundFullyFilled`:

```
bool roundFilled = soldSteps >= fraction.minSharesToRaise;
if (roundFilled) {
    revert CannotClaimRefundWhenRoundFullyFilled();
}
```

Similarly, `closeFraction` blocks manual close once the threshold is reached, but the error is named `CannotCloseAFullRound`:

```
if (fraction.soldSteps >= fraction.minSharesToRaise) {
    revert CannotCloseAFullRound();
}
```

Elsewhere, "full" corresponds to an actually sold-out round, e.g., `RoundFilled` is emitted only when `newFractionsSold == totalSteps`. This naming mismatch can mislead integrators and operators into conflating "threshold reached" with "fully sold.".

Recommendation: Rename the two "full" errors to reflect threshold semantics, for example `CannotClaimRefundWhenThresholdReached` and `CannotCloseWhenThresholdReached` and update NatSpec/comments accordingly.

Glow: Fixed in commit 92d5166.

Cantina Managed: Fix verified.

3.4.9 Unnecessary CFH resolution on non-payout paths

Severity: Informational

Context: OffchainFractions.sol#L348-L353

Description: The contract resolves the recipient's counterfactual holder (CFH) address on every buy inside `_calculatePurchaseDetails`, even when the purchase is below the minimum threshold and no payout to the recipient occurs. This incurs an external call and hashing work that are not used on escrow-only paths.

```
{
    address toInStruct = fraction.to;
    details.sendTo = fraction.useCounterfactualAddress
        ? i_CFHFactory.getCurrentCFH({user: toInStruct, token: fraction.token})
        : toInStruct;
}
```

Below the threshold, funds are held in the contract, so resolving `sendTo` there is unnecessary. Resolving once and caching across calls is unsafe because `getCurrentCFH` can legitimately change if the recipient (or an approved operator) advances their salt via the `CounterfactualHolderFactory`.

Recommendation: Resolve the recipient (or CFH) lazily only when actually sending to the recipient (i.e., in the post-threshold branches of `_handlePurchaseTransfers`) and compute it fresh each time. Consider removing `sendTo` from `PurchaseDetails` to avoid accidental early resolution.

Glow: Acknowledged.

Cantina Managed: Acknowledged.

3.4.10 `buyFractions` ignores `creditTo` for refunds/ownership lookup

Severity: Informational

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

Description: `OffchainFractions.buyFractions` accepts a `creditTo` address so the caller can buy on behalf of someone else. The value is only used when emitting `FractionSold`, but the contract stores the buyer's allocation in `stepsPurchased[msg.sender][creator][id]`. Later, `claimRefund` requires the original buyer address to match the stored slot. If a purchase was made with `creditTo != msg.sender`, the credited party can see their allocation off-chain (via the event) but cannot claim refunds or interact with their position on-chain because every lookup key is anchored to `msg.sender`.

Recommendation: Merely informative. Consider documenting this behaviour. The caller(`msg.sender`) will keep the responsability of claiming any future refunds, not the `creditTo` address.

Glow: Acknowledged.

Cantina Managed: Acknowledged.

3.4.11 Shared CFH escrow queue across multiple sales on same token

Severity: Informational

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

Description: When `useCounterfactualAddress` is enabled, every fraction sale deposits into `factory.getCurrentCFH(address(this), token)`, meaning multiple sales on the same token share a single "current" CFH. While it's natural to worry that triggering one sale's payout would strand funds earmarked for other sales, `_execute` handles this safely: before deploying the CFH that executes the payout, it records the next CFH address in transient storage. The deployed holder relays the payout and then forwards any residual balance to that next address before the factory increments `nextSalt`. Consequently `getCurrentCFH(address(this), token)` always points to the address holding the remaining escrow, and no leftovers are lost. This behavior is intentional but might surprise integrators who expect per-sale isolation.

Recommendation: Document the shared CFH queue semantics (e.g., via NatSpec or developer docs) so downstream teams understand why multiple sales on the same token can safely coexist. No code changes required.

Glow: Fixed in commit f2bbc24.

Cantina Managed: Fix verified.

3.4.12 Clarify pre-threshold escrow destination when CFH is enabled

Severity: Informational

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

Description: `_handlePurchaseTransfers`' comments and NatSpec describe "All fundraised amounts before `minSharesToRaise` is reached are held in the contract." before the minimum is reached, but when `useCounterfactualAddress == true` the logic actually routes deposits into the contract's current CFH (`factory.getCurrentCFH(address(this), token)`). Operators relying on the comments might monitor the `OffchainFractions` contract and miss that the escrow sits in a CFH instead. The implementation is correct, the documentation just lags behind and could cause minor operational confusion.

Recommendation: Update the NatSpec/comments around `_handlePurchaseTransfers` to explain that, with CFH enabled, pre-threshold funds live in the `OffchainFractions` CFH address rather than the contract balance. That would keep any on-chain monitoring aligned with actual behaviour.

Glow: Fixed in commit 18bb3ad.

Cantina Managed: Fix verified.

3.4.13 Remove lingering `@auditor` comment

Severity: Informational

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

Description: Inside `claimRefund` we still ship the inline note `/// @auditor - Let me know if you think we can remove this, I don't think it's necessary` above the line that decrements `fraction.soldSteps`. That comment suggests the code might be optional even though the state update is needed to keep `soldSteps` accurate after refunds.

Recommendation: Delete the comment and leave the decrement in place. The logic should remain as-is, but the source should no longer imply it's removable.

Glow: Fixed in commit 7267ecc.

Cantina Managed: Fix verified.

3.4.14 Limit sale expiration to avoid indefinite capital lock-up

Severity: Informational

Context: OffchainFractions.sol#L158

Description: `_validateFractionCreationParams` only checks that `expiration > block.timestamp`. A creator can therefore set expiration years into the future while also choosing a `minSharesToRaise` close to `totalSteps` with a very high amount as `totalSteps`. Until either the minimum is hit or the clock passes expiration, buyer deposits stay escrowed in the contract/CFH. With no upper bound, a malicious or misconfigured sale can freeze capital for an arbitrarily long time.

Recommendation: Enforce a reasonable ceiling on expiration (for example `block.timestamp + MAX_DURATION`, where `MAX_DURATION` is a constant or governance-controlled value) so raises can't lock funds indefinitely.

Glow: Fixed in commit f5524e7.

Cantina Managed: Fix verified.