SHERLOCK

Security Review For Ticker Spring

Collaborative Audit Prepared For: **Ticker Spring**
Lead Security Expert(s): **ctf_sec**
hildingr
Date Audited: **October 5 - October 14, 2026**

Introduction

TickerSpring vaults are automated USDG liquidity vaults on Robinhood Chain. Each vault manages a concentrated liquidity position pairing USDG with a tokenized stock or crypto asset on Uniswap, Up or Ramses, earning trading fees and gauge rewards within loss limits enforced on-chain.

The audit covers:

- Vault: the share token and its accounting, with keeper-filled deposit requests, pro-rata redemptions in kind or in USDG, rebalancing within loss limits, fee reserves, pause and recovery.
- Venues: hold each vault's single position in a Uniswap V3, Uniswap v4 (hookless), Up or Ramses V3 pool, including gauge rewards on Up and Ramses.
- PriceModule and ChainedPriceModule: price the pool token and gauge rewards in USDG from on-chain TWAPs, cross-checked against price feeds and reference pools where configured, and refuse to price a market that is moving too fast.
- KyberSwapper: the only contract that swaps, limited to a pinned KyberSwap router and executor.
- RedeemQueue: lets a holder queue a redemption that a pool briefly refuses; the keeper completes it shortly after.
- RefundEscrow: holds a deposit refund the vault cannot deliver, payable only to the depositor it belongs to.

More about TickerSpring: https://tickerspring.com/docs

Scope

Repository: Polypup/tickerspring-audit

Audited Commit: 17fb57852019f818a6edb687ff256a9072a94bc7

Final Commit: cac320cbc26050b8eb0ec4b5f67782aff00e2631

Files:

- vaults/src/ChainedPriceModule.sol
- vaults/src/interfaces/IKyberSwapper.sol
- vaults/src/interfaces/IPriceModule.sol
- vaults/src/interfaces/IVenue.sol
- vaults/src/KyberSwapper.sol
- vaults/src/PriceModule.sol
- vaults/src/RedeemQueue.sol

- vaults/src/Vault.sol
- vaults/src/venues/NpmVenue.sol
- vaults/src/venues/RamsesVenue.sol
- vaults/src/venues/UpVenue.sol
- vaults/src/venues/V3Venue.sol
- vaults/src/venues/V4Venue.sol

Final Commit Hash

cac320cbc26050b8eb0ec4b5f67782aff00e2631

Findings

Each issue has an assigned severity:

- High issues are directly exploitable security vulnerabilities that need to be fixed.
- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- Low/Info issues are non-exploitable, informational findings that do not pose a security risk or impact the system's integrity. These issues are typically cosmetic or related to compliance requirements, and are not considered a priority for remediation.

Issues Found

High	Medium	Low/Info
1	3	0

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue H-1: RamsesVenue reads the global anti-sybil window instead of the gauge's, so vault rewards can be forfeited to Ramses [RESOLVED]

Source: https://github.com/sherlock-audit2/2026-10-ticker-spring-oct-5th-2026/issues/17

Summary

`RamsesVenue` predicts whether Ramses will pay a reward claim by calling Ramses' reward validator itself. Since Ramses' upgrade of 2026-10-05, the validator reads the anti-sybil window of its caller, so the venue always gets the global 99 s window and never a gauge's own. Once Ramses gives the vault's gauge a longer window, the venue claims and changes liquidity while the gauge withholds, and the vault's rewards are forfeited to Ramses.

Vulnerability Detail

Ramses upgraded the RewardValidator, the Voter, the gauge beacon and the AccessHub in one transaction (tx, block 81,136,195). The team's Ramses fork tests pin block 77,390,000 (Oct 1), before it (RamsesVenueFork.t.sol#L157). The validator's source is not published; a decompilation that matches the live contract on a fork is here.

1. `RamsesVenue._withheld` calls `validateReward` directly (RamsesVenue.sol#L335-L337), so inside the validator `msg.sender` is the venue.
2. The validator reads `voter.timeThresholdForGauge(msg.sender)` (decompiled L220). The venue has no window of its own, so the Voter returns the global `timeThresholdForRewarder`, 99 s (Voter.sol#L1009-L1013).
3. When the gauge checks a claim, it calls the validator itself (GaugeV3.sol#L535-L551), so it gets the gauge's own window, set by a Ramses operator with no timelock (AccessHub.sol#L314-L316).

With a gauge window W above 99 s, the venue sees ”clear” from 99 s after a liquidity change while the gauge withholds until W. In that gap:

- `claimRewards` claims (RamsesVenue.sol#L236-L251), and every harvest calls it (Vault.sol#L1172-L1190).
- `canChange` lets `add` and `remove` through (RamsesVenue.sol#L221-L229), and Ramses' position manager claims on each change.

The gauge records each such claim as forfeited (GaugeV3.sol#L468-L481), and anyone can sweep it to Ramses' treasury (GaugeV3.sol#L511-L527).

No gauge has its own window yet; the issue starts once Ramses sets one.

Impact

Holders lose the vault's gauge rewards (RAM, plus any pool-token gauge rewards), which are its main yield, and the protocol loses its 30% fee share on them. Any harvest or liquidity change inside the gap forfeits everything accrued since the last paid claim. Anyone can trigger it with a zero-share `redeemInKind`, and an attacker holding 1 USDG of shares who redeems a sliver every ~100 s keeps the vault from receiving any RAM. Principal is not affected.

Recommendation

In `_withheld`, also hold for the gauge's own window, and treat a reverting validator as withheld, since the gauge does:

```
         ) returns (bool w) {
-            withheld = w;
-        } catch { }
+            if (w) return true;
+        } catch {
+            return true;
+        }
+        try VOTER.timeThresholdForGauge(gauge) returns (uint256 window) {
+            return block.timestamp - changedAt <= window;
+        } catch {
+            return false;
+        }
```

(`IRamsesVoter` needs `timeThresholdForGauge(address)`, and `_withheld` returns `bool` directly.)

We still call `IRamsesRewardValidator(validator).validateReward` for its blacklist checks (owner, receiver, origin and position), which do not depend on the caller. The validator has no public getter for its wallet blacklist, so the venue cannot check it any other way. That call still uses the default window, so the venue waits for whichever window is longer. So we will have this trade-off:

- Gauge window longer than the default: the added check holds the venue until the gauge's window passes, so nothing is forfeited.
- Gauge window shorter than the default: the venue still waits the default, as it does today. We have an extra delay.

Issue M-1: Cash redemptions can be permissionlessly converted to in-kind redemptions with no slippage protection [RESOLVED]

Source: https://github.com/sherlock-audit2/2026-10-ticker-spring-oct-5th-2026/issues/15

Summary

After `GRACE`, any address can convert a queued USDG redemption into an in-kind redemption, bypassing the selected settlement mode and stored `minAssets` and let user suffer from slippage protection.

Vulnerability Detail

The normal keeper path forwards the stored `minAssets` for a cash redemption (RedeemQueue.sol#L188). However, `settleInKind()` is open to any caller (RedeemQueue.sol#L199), and its eligibility check does not require `r.inKind` to be true (RedeemQueue.sol#L204).

It instead calls `redeemInKind()` with both output minimums set to zero (RedeemQueue.sol#L205), making the Vault's minimum-output check ineffective (Vault.sol#L632).

Although GRACE is documented as ensuring that no request depends on the keeper for longer than ten minutes, an unrelated caller can finalize the request under materially different terms.

Impact

The caller cannot redirect the assets, but the receiver can be forced to accept pool and reward tokens instead of the requested USDG amount and make user suffer from slippage loss.

Code Snippet

- Cash settlement uses the requester's stored `minAssets`: RedeemQueue.sol#L188
- Permissionless `settleInKind()` entry point: RedeemQueue.sol#L199
- Eligibility check omitting `r.inKind`: RedeemQueue.sol#L204
- Forced in-kind call with zero minimums: RedeemQueue.sol#L205
- Minimum-output check neutralized by the zero arguments: Vault.sol#L632

Tool Used

Manual Review

Recommendation

Only allow `settleInKind()` for requests that originally selected in-kind settlement. Converting a USDG request should require explicit owner authorization and user-defined `minUsdg` and `minToken` limits.

Discussion

ctf-sec

Fix looks good. https://github.com/Polypup/tickerspring-audit/pull/3

Issue M-2: Permissionless claiming can prematurely finalize another user's recovery payout [RESOLVED]

Source: https://github.com/sherlock-audit2/2026-10-ticker-spring-oct-5th-2026/issues/16

Summary

Anyone can finalize another holder's blocked-token claim. Because the payout ratio uses the Vault's current balance, a third party can force a holder to claim before later recoveries or donations arrive.

Vulnerability Detail

`claimBlocked()` is callable by any address for an arbitrary `account` (Vault.sol#L845). It calculates the payout using the live ratio `_idle(blocked) / blockedUnpaid` (Vault.sol#L851), reduces the remaining denominator (Vault.sol#L853), and permanently deletes the claim (Vault.sol#L854).

A remaining claimant can settle another user's claim while backing is temporarily reduced or before a delayed venue residual arrives. NPM venues retry such residual transfers on later calls (NpmVenue.sol#L375); the V4 venue has equivalent behavior (V4Venue.sol#L304). Later inflows are therefore distributed only among claims that remain unpaid.

If the balance never changes, permissionless execution has no economic effect.

Impact

A third party can choose when another holder prematurely a recovery haircut or loses participation in subsequent inflows. Assets cannot be redirected, but later recoveries may be redistributed toward the remaining claimants.

If post-snapshot inflows are intentionally excluded from already-paid claims, this is a documented design trade-off rather than direct fund loss.

POC

https://gist.github.com/JeffCX/6bd0ced76e6ef75b7753aba7f91e0e46

Code Snippet

- Permissionless arbitrary-account entry point: Vault.sol#L845

- Payout calculated from the live Vault balance: Vault.sol#L851
- Remaining-claim denominator reduced: Vault.sol#L853
- Settled claim permanently deleted: Vault.sol#L854
- Payout cannot be redirected away from the claim holder: Vault.sol#L855

Tool Used

Manual Review

Recommendation

Restrict claiming to `account` or an approved operator. If permissionless settlement is required, finalize and freeze the recovery conversion rate before third parties may execute claims.

Discussion

ctf-sec

Fix looks good https://github.com/Polypup/tickerspring-audit/pull/4

Issue M-3: One frozen depositor keeps deposits closed for good after an issuer burn into the pending USDG [RESOLVED]

Source: https://github.com/sherlock-audit2/2026-10-ticker-spring-oct-5th-2026/issues/18

Summary

After an issuer burn reaches the pending USDG, deposits stay closed until every open request has been refunded. If USDG has frozen one request owner, that refund can never be sent, so the vault cannot take deposits again.

Vulnerability Detail

1. `_touchShortfall` records the shortfall (`coverNum, coverDen`) once the USDG balance is below what is held for the requests (Vault.sol#L1055-L1058).
2. While `coverDen != 0`, `_requireDepositsOpen` reverts `Closed` (Vault.sol#L1040-L1045), and `settleDeposits` only refunds, then returns (Vault.sol#L522-L528).
3. `coverDen` is reset in only one place: in `_refund`, when `pendingDeposits` reaches 0 (Vault.sol#L1089). So every open request has to be refunded to its owner.
4. `_refund` sends USDG to `r.owner` and returns false if the transfer fails, leaving the request open (Vault.sol#L1086). USDG (Paxos) reverts any transfer to a frozen address. `cancelDeposit` goes through the same `_refund` and reverts `Closed` (Vault.sol#L494).

No other path removes a request: the fill path is behind the `coverDen` check, and no owner or guardian function can evict a request or change who it pays. A frozen owner's request therefore keeps `pendingDeposits != 0` and `coverDen != 0` for as long as the freeze lasts.

The code means to prevent this. `settleDeposits` says the keeper refunds every request "so one owner who never cancels cannot keep deposits closed" (Vault.sol#L518-L520), but that does not hold for an owner who cannot receive USDG. Without a burn, the same frozen request is just skipped and deposits stay open (control test below).

Impact

The vault stops taking deposits for as long as one request owner stays frozen, which for a sanctioned address usually means for good. `requestDeposit` and every `settleDeposits` fill revert or return early, and neither the owner nor the keeper can undo it. The vault has

to be replaced. The burned depositors' remaining USDG (other than the frozen owner's) can still be refunded, and holders can still redeem.

It needs two issuer actions: a burn that reaches the pending USDG (a case the vault is built to handle) and a freeze on one depositor while their request is open.

Recommendation

When a refund under a haircut cannot be sent, remove the request anyway and owe its amount to the owner, to be claimed later. Keep the owed USDG out of the holders' USDG (`balance - pendingHeld()` at Vault.sol#L1242 and in `_touchShortfall`).

Alternatively, send a refund that cannot be delivered to a separate escrow contract that pays only the owner, once unfrozen. The USDG then leaves the vault's balance, so no other accounting changes.

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.