IRONNODE

ironnode.io

FINAL REPORT

OCELOT

Solana Program Security Audit

Prepared for Ocelot

IronNode Security

17 November 2026 / Version 1.6

IRONNODE

01

SECTION 01

Executive summary

1 Executive Summary

1.1 Introduction

Ocelot is an Anchor-based Solana leveraged vault. Holders receive LT shares against USDC-backed market positions. The program manages Phoenix trader exposure, share issuance and redemption, fees, withdrawal capacity, authority handoffs and emergency recovery.

IronNode reviewed the agreed source snapshot, the two contract-facing transaction builders and supporting build and integration material. The assessment records 13 findings: two Medium, four Low and seven Informational. No finding is classified Critical or High. All 13 findings are solved in the reviewed source stack. The final recheck includes the V3 retirement in PR #25 and the market-maker wallet policy feature in PR #24 at the pinned revision below.

1.2 Scope and revision

Item	Reviewed scope
Client	Ocelot
Auditors	Alp Onaran and Kerem Solmaz, IronNode Security
Review period	27 October - 9 November 2026
Repository	nik-suri/ocelot-contract-audit
Controlling audit commit	4ef4ee44f78ec76cb813ead5ca1093f7c5dfc69c
Remediation review commit	c150589dceab39d7c786505fdc5c2be8807a29c7
V3 retirement commit	cd7aa8279b318009a9834c434b96275025f0b7fc
Original source revision	d226f4d28cc6060b5620c937c9ddfdc2c04c5a50
Declared program	CpLah6JAC9967YiVQnQJegfCKEjLEcUYmx3PCYgg9xWf
Runtime scope	17 Rust contract files and 2 TypeScript transaction builders
Functional lines	6,330 Rust; 383 TypeScript; 6,713 total
Interface and build inputs	8 files reviewed as contract context

The controlling baseline is the agreed 4ef4ee44 commit label. All 18 on-chain Rust source hashes in the saved audit inventory match the supplied archive, including one test-only model outside the 17-file runtime scope. Functional lines exclude blank, comment, delimiter-only and test-only lines. The exact 19 runtime files are listed below.

Contract path under src, or transaction builder	Functional lines

Contract path under src, or transaction builder	Functional lines
error.rs	109
events.rs	181
finance.rs	328
finance_actions.rs	742
finance_admin.rs	303
finance_custody.rs	627
finance_state.rs	89
finance_user.rs	192
finance_view.rs	223
flows.rs	929
global_throttle.rs	64
lib.rs	1879
state.rs	132
streaming.rs	202
venue/nav.rs	35
venue/phoenix.rs	288
venue.rs	7
packages/sdk/src/finance-instructions.ts	287
packages/web/src/lib/phoenix-instruction-data.ts	96

The contract paths above are under packages/leveraged_vault/programs/leveraged_vault/src/. The primary TypeScript targets are the SDK finance builder and web instruction-data encoder. Broader keeper services, website, API, infrastructure, signing-key custody, and independent Phoenix or Ember audits were outside the agreed scope. The reference program binary was not supplied.

1.3 Findings overview

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	2	4	7

13 **FINDINGS**

13 solved in reviewed source.

The Medium findings concern negative-decimal Phoenix valuation that overissues shares and a one-raw-unit LT split that blocks a near-total exit. The Low findings cover unauthorized market and child admission, an order-price guard, stressed partial exits and keeper handoff. In the reviewed source, V3 redemption is retired and V4 supports the tested one-unit and stressed partial exits; the remaining paths were addressed in earlier PRs.

The seven Informational findings are solved in the reviewed source: V0 economic entrypoints are retired; post-order marking, Phoenix key decoding, donated-collateral cleanup and adverse-redemption fee protection are updated. Additional recovery results appear in Section 1.7 and are not counted as new numbered findings.

No.	Security analysis	Risk	Status
1	Negative-decimal Phoenix pricing can overissue vault shares	Medium	Solved
2	One raw LT unit can block a near-total withdrawal	Medium	Solved
3	Unrestricted market creation can reserve a Phoenix child	Low	Solved
4	Rounding can weaken the shared order price bound	Low	Solved
5	A stressed partial redemption can fail the post-exit margin check	Low	Solved
6	Authority transfer leaves the outgoing key with keeper access	Low	Solved
7	Legacy emergency de-risking can accept an adverse execution	Informational	Solved
8	Legacy minting can issue shares despite an adverse fill	Informational	Solved
9	Legacy floored NAV can overissue shares in an extreme state	Informational	Solved
10	Post-order valuation can reuse the pre-order mark	Informational	Solved
11	High bits in a Phoenix position key can block marking	Informational	Solved
12	A non-position-only trader map can block finance marking	Informational	Solved
13	Accrued fee allocation can be impaired after an adverse redemption	Informational	Solved

1.4 Review approach and test record

The contract review traced market admission, Phoenix child assignment, authority and keeper roles, minting, redemption, NAV and share rounding, streamed fees, custody, finance policy, shared withdrawal capacity, emergency reduction, recovery, migration and release state. The two transaction builders were checked for instruction data and account construction at their contract boundary.

Compiled local tests used synthetic Phoenix and Ember accounts. Stock venue tests were kept separate from tests that deliberately changed the venue to produce adverse or favorable fills, or fresh trader registration. Negative controls challenged whether the same effect survived changed ticks, marks, lot sizes, roles, fee policies or account state.

Recorded outcome	Cases
Passed local assertions or test cases	131
Completed model or source checks	36
Failed challenge assertions	11
Blocked by stock mock capability	1
Total registered IronNode cases	179

Ten failed assertions were falsification controls: the stock venue did not reproduce behavior that required a modified venue. The remaining failed assertion exposed the baseline shared-keeper handoff. The blocked case was a fresh-child registration path unsupported by the stock mock. The register is a case ledger, not a claim that 179 tests passed.

Additional validation	Recorded result
Baseline host Rust suite	125 tests passed in the saved audit run
Keeper PR #10 replay	Shared, independent and wrong-signer controls passed; 117 host Rust tests passed
Formal setup	38 Z3 rules; 16 guard-removal sensitivity controls
PR #11 follow-up host suite	122 of 122 Rust tests passed in the reviewed source snapshot
PR #23 local source and venue checks	129 Rust; 363 controlled venue; 26 real-venue composition; 8 direct Phoenix tests passed
PR #25 local retirement review	Rebuilt ELF matched the manifest; 129 Rust, 359 controlled venue and 225 SDK/support tests passed
PR #24 wallet policy review	131 Rust; 388 controlled venue; 28 real Phoenix/Ember; 49 focused SDK/ABI checks passed

These follow-up suites are separate from the 179-case baseline ledger.

1.5 Formal verification results

The maintained Z3 setup ran 38 source-derived arithmetic and transition rules. Under their encoded premises, 32 violation queries had no counterexample and six returned expected counterexamples. The six correspond to the near-total exit, default keeper handoff, open market admission, possible in-call mark change, low-price bound and negative-decimal valuation. Sixteen deliberate guard removals were detected by sensitivity controls.

An unsatisfiable query establishes a property only in its model. A satisfying model is not by itself an executable Solana transaction. The rules do not constitute a machine proof of the full Rust program, SBF binary, Anchor account checks or Phoenix and Ember handlers.

Rule	Security property	Result
MINT-01	V1 share issuance does not dilute existing shares when credited value is accurate	No violating model
MINT-02	Floor issuance loses less than one raw share of entitlement	No violating model
EXIT-01	Pro-rata withdrawal capacity cannot exceed pre-exit equity	No violating model
EXIT-02	A successful partial exit retains the protected equity	No violating model
EXIT-03	Computed close lots never exceed the position	No violating model
EXIT-04	A near-total withdrawal can require zero close lots	Counterexample in model
FEE-01	Redemption retires exactly the allocated fee liability	No violating model
FEE-02	New streaming accrual is capped by positive net assets	No violating model
THROTTLE-01	Successful consumption cannot create negative or excess capacity	No violating model
THROTTLE-02	Enabled refill is monotone and capped at the configured limit	No violating model
THROTTLE-03	Raising a policy limit does not itself issue new credit	No violating model
THROTTLE-04	Standalone fee settlement preserves configured user headroom	No violating model
RECOVERY-01	Separated net and fee claim floors do not exceed gross pro-rata entitlement	No violating model
RECOVERY-02	Campaign consumption stays within captured shares and loss allowance	No violating model
RISK-01	A permitted reduction strictly improves the exposure-to-equity ratio	No violating model
AUTH-01	Authority transfer automatically revokes a default outgoing keeper	Counterexample in model
AUTH-02	Only an approved operator can create a new market ID	Counterexample in model
MIGRATION-01	A completed V0-to-V1 transition cannot be replayed	No violating model

Rule	Security property	Result
CASH-01	Unsolicited cash-vault donations do not raise recognized holder backing	No violating model
POLICY-01	A stale market-finance revision cannot authorize a policy update	No violating model
PRICE-01	A V1 buy limit cannot exceed the exact basis-point cap	No violating model
PRICE-02	A V1 sell limit cannot fall below the exact basis-point floor	No violating model
MARK-01	An unchanged funding index guarantees an unchanged mark	Counterexample in model
CASH-02	A recognized receipt cannot exceed the token vault balance	No violating model
CASH-03	A recognized payment cannot spend donated surplus as holder cash	No violating model
CASH-04	Positive cash cannot make an insolvent venue snapshot acceptable	No violating model
FEE-03	Redemption proceeds have one complete and nonnegative destination	No violating model
FEE-04	A one-day streaming accrual carries its fractional remainder exactly	No violating model
FEE-05	A capped streaming accrual does not carry phantom fee remainder	No violating model
RECOVERY-03	Partial recovery leaves nonclaimants' pre-execution gross claim intact	No violating model
RECOVERY-04	A full recovery distributes the post-execution gross exactly	No violating model
RECOVERY-05	Campaign rollover cannot regrant loss budget on old residual claims	No violating model
THROTTLE-05	A redemption is charged for the larger of quoted and realized outflow	No violating model
MINT-03	A successful V1 mint cannot pass more than one-basis-point rounding loss	No violating model
SUPPLY-01	Honest mint and burn transitions preserve abandoned-share gap	No violating model
CONTRIB-01	A cash contribution cannot credit holders beyond its value after fee repair	No violating model
PRICE-03	Shared maintenance sell guard enforces an exact ten-percent price floor	Counterexample in model
VALUE-01	Negative-decimal short valuation never adds phantom equity	Counterexample in model

The executable assumptions, source checks, rules and saved results are in the accompanying Ocelot security test package. The reported counterexamples were reconciled with compiled tests and the conditional classifications above.

1.6 Evidence boundaries and reproduction

The original assessment is tied to the supplied source snapshot and local evidence. The PR #23 follow-up added local transactions with candidate Ocelot, Phoenix and Ember binaries. The PR #25 review rebuilt the new Ocelot program and reran controlled-venue, Rust and SDK tests. The PR #24 review then covered the wallet policy diff on top of PR #25 and ran local Rust, controlled-venue, real Phoenix/Ember and SDK tests. None of these reviews replays live funded markets. A 14 November read-only inventory lists 28 V1 markets and no V0 markets. Stock and modified venue tests remain identified separately in each original finding.

The separate PR #11 rewards review compared the new CPI account layouts with the pinned public Rise source, checked additive interface changes and verified 313 packaged hashes. No new source-level finding survived that review. The new instruction was not part of the 4ef4ee44 baseline; its deployed venue compatibility and immutable USDC destination still require release checks.

The accompanying Ocelot-security-tests-4ef4ee4.zip contains 41 integration files, 13 host Rust modules, a legacy NAV check, three Python checks, 38 current SMT rules, earlier models and challenge patches. The package README explains the disposable-copy setup and identifies which tests require modified venue mocks. Production signing keys are not included.

Selected code and test excerpts appear in the findings; they are not stand-alone scripts. The full tests can be reproduced with the source archive, local build toolchain and commands in the package README. The formal runner checks the source manifest before solving:

```
Formal test runners

python3 formal/run.py --source-root "$OCELOT_SOURCE_ROOT"
python3 formal/negative_controls.py
```

1.7 Additional review observations

These source-level recovery results are separate from the 13 numbered findings and their severity totals.

Observation	Remediation status
Cash-only exit	Solved. A flat child backed by accounted cash can redeem without requiring a fresh Phoenix mark.
Fee-only terminal state	Solved. Recovery pays the remaining accrued fee and clears the abandoned accounting share atomically, allowing the market to reopen.
Negative venue equity	Solved. Existing accounted cash can restore positive Phoenix equity, followed by bounded reduce-only maintenance. Long and short local real-venue cases passed.

Authority and operational state

PR #4 enforces governed market and Phoenix child admission. The shared withdrawal capacity retains Ocelot's intended first-come-first-served behavior.

PR #10 transfers keeper alongside authority when the outgoing key holds both roles and preserves an independently appointed keeper.

1.8 Remediation and decisions

This final recheck reviews PR #24 at commit c150589dceab39d7c786505fdc5c2be8807a29c7, stacked above PR #25 at cd7aa8279b318009a9834c434b96275025f0b7fc. All 13 findings are Solved in the reviewed source. Section 2 retains the original baseline findings and proof.

Reviewed path	Status	Short explanation
Issue #2: one-unit exit	Solved	V3 redemption rejects before economic execution; V4 pays the tested split and near-total exits.
Issue #5: stressed partial exit	Solved	V3 redemption rejects before economic execution; V4 permits the tested non-worsening reductions.
Issues #7-9: legacy V0 economics	Solved	Five legacy economic entrypoints now stop before venue execution or share and NAV changes.
Issue #10: post-order mark	Solved	Share credit is separated from mark movement; NAV and margin use the observed post-order mark.
Issue #11: Phoenix position key	Solved	The decoder matches the lower-32-bit asset identity and supported storage discriminants.
Issue #12: donated child entry	Solved	The parser recognizes collateral entries; cleanup and mint succeed atomically with real Phoenix and Ember locally.
Issue #13: fee allocation	Solved	User exits must preserve the allocated streaming fee and satisfy price and loss bounds.
Fee-only terminal state	Solved	Recovery pays the final fee and clears terminal accounting in one transaction.
Negative venue equity	Solved	Accounted cash can repair equity before bounded reduce-only maintenance.

PR #23 validation passed 129 Rust tests, 363 controlled-venue tests, 26 full Ocelot/Phoenix/Ember composition tests and eight direct Phoenix tests. The PR #25 mainnet-profile rebuild matched its submitted ELF; 129 Rust, 359 controlled-venue and 225 SDK/support tests passed. The PR #24 combined candidate passed 131 Rust, 388 controlled-venue, 28 local real Phoenix/Ember and 49 focused SDK/ABI checks. Its independently rebuilt ELF and IDL matched the recorded hashes.

The PR #25 V3 handler now returns RedemptionV3Retired without running the old redemption path. The SDK V3 builder rejects locally and the shared web/CLI encoder selects V4 for finance-active markets. The PR #24 wallet policy feature was reviewed without an additional reportable finding. Old V3 clients must use V4 after the program upgrade.

Solved describes the reviewed source. The deployed program, live policy accounts and production client routing were not matched to this revision.

IRONNODE

02

SECTION 02

Findings

2 Findings

2.1 Negative-decimal Phoenix pricing can overissue vault shares

MEDIUM SOLVED

A stock-mock compiled-program round trip demonstrates local holder dilution. Production reachability remains unverified.

In a Phoenix short market with negative base-lot decimals, Ocelot divides native tick value before valuing the position. A nondivisible tick loses its remainder. The signed-notional calculation then multiplies the truncated mark back by the scale, understating the short liability. The V3 mint path can count that phantom equity as a new minter's contribution and issue too many shares.

2.1.1 Code location

packages/leveraged_vault/programs/leveraged_vault/src/lib.rs:1502–1511

```
    } else {
        let scale = 10u128
            .checked_pow((-base_lot_decimals) as u32)
            .ok_or(LtError::Overflow)?;
        (price_ticks as u128)
            .checked_mul(tick_size as u128)
            .ok_or(LtError::Overflow)?
            .checked_div(scale)
            .ok_or(LtError::Overflow)?
    };
```

packages/leveraged_vault/programs/leveraged_vault/src/finance_view.rs:41–53

```
pub fn signed_notional(base: i64, mark: u64, decimals: i8) -> Result<i128> {
    let scale = 10i128
        .checked_pow(decimals.unsigned_abs() as u32)
        .ok_or(LtError::Overflow)?;
    let n = (base as i128)
        .checked_mul(mark as i128)
        .ok_or(LtError::Overflow)?;
    if decimals >= 0 {
        Ok(n / scale)
    } else {
        n.checked_mul(scale).ok_or(error!(LtError::Overflow))
    }
}
```

2.1.2 Impact

In the compiled stock-mock fixture, a 10 USDC mint and redemption returned 14.072499 USDC. Incumbent true equity fell from 100 to 95.875157 USDC. A divisible-tick control returned 9.960030 USDC and preserved the incumbent's 100 USDC. This is local value-transfer evidence, not measured deployed profit.

2.1.3 Recommendation

To mitigate this issue, the following improvements are recommended: preserve the exact native quote value through short-position valuation and share accounting, using checked wide integers before converting units. Until corrected, reject configurations whose tick value is not divisible by the negative-decimal scale and review any active affected market.

2.1.4 Remediation status

SOLVED: Client fixed this issue in commit 8090fbe7 by preserving native quote-per-lot precision through valuation and share calculations.

2.1.5 Proof of concept

The compiled Ocelot program ran through the repository LiteSVM fixture with synthetic Phoenix-owned accounts and the unmodified venue mock. Six local cases passed, including divisible-tick and low-price controls. A separate adverse-fill stress case required a modified mock. No currently funded, fillable negative-decimal market or deployed-binary equivalence has been established.

```
Ocelot-security-tests-4ef4ee4/integration/stock/negative-decimal-round-trip.test.ts:100-109

  it('redeems excess shares for a modeled cash gain at existing holders expense', async () => {
    const result = await redeemMinted(19n);
    assert.equal(result.userReceived, 14_072_499n);
    assert.equal(result.incumbentTrueEquity, 95_875_157n);
  });

  it('divisible tick negative control returns less than the 10 USDC paid to mint', async () => {
    const result = await redeemMinted(20n);
    assert.equal(result.userReceived, 9_960_030n);
    assert.equal(result.incumbentTrueEquity, 100n * MICRO);
```

2.2 One raw LT unit can block a near-total withdrawal

MEDIUM SOLVED

The baseline one-unit split was reproduced locally. In PR #25, V3 returns a retirement error without a venue CPI, and V4 passes the tested exit.

A holder can retain one raw LT unit while another holder owns 99,999,999 units of a one-lot market. Redemption floors the large holder's proportional close to zero lots, then rejects the exit before any Phoenix order. The affected holder may be a later recipient rather than the original minter.

2.2.1 Code location

packages/leveraged_vault/programs/leveraged_vault/src/streaming.rs:157–165

```
pub fn redemption_lots(lots: u64, shares: u64, accounting: u64, actual: u64) -> Result<u64> {
    require!(actual <= accounting, LtError::ShareSupplyMismatch);
    require!(shares > 0 && shares <= actual, LtError::ZeroAmount);
    let floor = fraction(lots, shares, accounting)?;
    // Leave at most the abandoned fraction of exposure, not a whole lot on dust backing.
    // This changes execution size by less than one lot, never the user's cash entitlement.
    let round_up = shares == actual && (lots as u128 * shares as u128) % accounting as u128 != 0;
    floor.checked_add(u64::from(round_up)).ok_or(error!(LtError::Overflow))
}
```

packages/leveraged_vault/programs/leveraged_vault/src/lib.rs:1663–1672

```
fn phoenix_validate_redeem_order_args(
    cfg: &PhoenixConfig,
    expected_base_lots: u64,
    before_base: i64,
    mark_price: u64,
    place_market_order_args: &[u8],
) -> Result<()> {
    // No cash-only bypass while exposure remains. Splitting claims cannot bypass this minimum.
    require!(expected_base_lots > 0, LtError::TradeBelowMinimumLot);
    require!(
```

2.2.2 Impact

A compiled stock-mock V1 first mint opened one long lot against 100 USDC net backing and issued 100,000,000 raw LT units. After one unit moved to another holder, the near-total owner could not redeem normally. Ten USDC of accounted cash did not bypass the gate. An authority policy change followed by a valid emergency close restored the local exit, so this is a holder-withdrawal delay requiring intervention, not proven theft or permanent loss.

2.2.3 Recommendation

To mitigate this issue, the following improvements are recommended: provide a holder-controlled exit for a sub-lot proportional close, or an explicit safe cash-settlement path that does not leave excess exposure with the remaining holder. Regression tests should cover one raw unit against almost the full supply, long and short positions, accounted cash, and abandoned-share recovery.

2.2.4 Remediation status

SOLVED: Client fixed this issue in commit cd7aa827 by retiring V3 redemption before economic execution; V4 supports the tested one-unit and near-total exits.

2.2.5 Proof of concept

Seven compiled-vault cases passed, including long and short positions, a separate large holder, rollback, return-of-unit, a cash variant and an authority-led emergency recovery. A related recovery test showed the same lot floor can block a near-total abandoned-share claim until the position closes. No affected current market, real Phoenix fill or deployed executable match was proved.

```
Ocelot-security-tests-4ef4ee4/integration/stock/one-unit-withdrawal.test.ts:150-174

    it((short ? 'short' : 'long') + ': one transferred raw LT unit blocks a near-total exit, and
        return of that unit permits a full close', async () => {
      const { t, redeem, maintain, order } = await setup(short);
      assert.equal(t.supply(), SUPPLY);
      assert.equal(t.amount(t.userLt), SUPPLY);
      t.send(createTransferInstruction(t.userLt, t.keeperLt, t.user.publicKey, 1n));
      assert.equal(t.amount(t.userLt), SUPPLY - 1n);
      assert.equal(t.amount(t.keeperLt), 1n);

      const before = t.amount(t.userUsdc);
      const rejected = t.send(redeem(SUPPLY - 1n, order(0n)), 'TradeBelowMinimumLot');
      assert.ok(rejected.logs.some((line) => line.includes('TradeBelowMinimumLot')));
      assert.equal(t.amount(t.userUsdc), before);
      assert.equal(t.amount(t.userLt), SUPPLY - 1n);
      assert.equal(t.supply(), SUPPLY);

      t.send(redeem(SUPPLY - 1n, Buffer.alloc(0)), 'TradeBelowMinimumLot');
      t.send(maintain(0), 'NothingToSettle');
      t.send(maintain(1), 'NothingToSettle');
      t.send(maintain(2), 'RecoveryClaimInvalid');

      t.send(createTransferInstruction(t.keeperLt, t.userLt, t.keeper.publicKey, 1n));
      assert.equal(t.amount(t.userLt), SUPPLY);
      t.send(redeem(SUPPLY, order(1n)));
      assert.equal(t.supply(), 0n);
      assert.ok(t.amount(t.userUsdc) > before);
```

2.3 Unrestricted market creation can reserve a Phoenix child

LOW SOLVED

Ocelot confirmed that open market admission is unintended. The registration and shared-capacity effects were reproduced locally with stated venue limits.

Any wallet can create an unused market ID and become its authority. That authority can first-claim an unassigned Phoenix child through set_phoenix_config, preventing its intended market from assigning the same child. Ocelot confirmed that open market creation and child reservation are outside its intended design; this resolves the prior policy question.

2.3.1 Code location

packages/leveraged_vault/programs/leveraged_vault/src/lib.rs:216–220, 547–563, 2604–2609

```
        let m = &mut ctx.accounts.market;
        m.authority = ctx.accounts.authority.key();
        m.pending_authority = Pubkey::default();
        m.keeper = ctx.accounts.authority.key(); // keeper defaults to authority; settable later
        m.usdc_mint = ctx.accounts.usdc_mint.key();
// ...
        let assignment = &mut ctx.accounts.child_assignment;
        if assignment.market == Pubkey::default() {
            assignment.market = ctx.accounts.market.key();
            assignment.trader = trader;
            assignment.phoenix_authority = phoenix_authority;
            assignment.parent_trader = parent_trader;
            assignment.trader_pda_index = trader_pda_index;
            assignment.parent_subaccount_index = parent_subaccount_index;
            assignment.trader_subaccount_index = trader_subaccount_index;
            assignment.bump = ctx.bumps.child_assignment;
        } else {
            require_keys_eq!(
                assignment.market,
                ctx.accounts.market.key(),
                LtError::PhoenixAccounts
            );
            require_keys_eq!(assignment.trader, trader, LtError::PhoenixAccounts);
// ...
    #[account(mut)]
    pub authority: Signer<'info>,

    pub system_program: Program<'info, System>,
    pub token_program: Program<'info, Token>,
    pub rent: Sysvar<'info, Rent>,
```

2.3.2 Impact

The confirmed effect is unauthorized market and child reservation. In a compiled source-level sequence, two 10 USDC round trips by a new zero-Ocelot-fee market exhausted a shared 20 USDC withdrawal bucket and caused an otherwise valid 11 USDC exit from an existing market to fail. The attacker recovered its USDC and its market fee vault stayed empty. Real Phoenix child registration, fill costs, current deployment state and lasting denial were not established. The intended first-come throttle remains a separate policy choice.

2.3.3 Recommendation

To mitigate this issue, the following improvements are recommended: restrict market creation to approved operators and bind child reservation to the intended market before configuration. Check assignment accounts for funded children and test first claims, duplicate claims and already assigned children.

2.3.4 Remediation status

SOLVED: Client fixed this issue in commit 7fbebd33 by requiring the governed global admin for new market and Phoenix child admission.

2.3.5 Proof of concept

An unrelated signer created an unused market and reserved an unassigned child; a later intended claim failed, while a preassigned child blocked the rogue claim. A separate compiled stock-mock test created and activated a new market with a pre-existing synthetic Phoenix child, then repeated a 10 USDC mint and redemption twice. The bucket fell from 20 to zero; an 11 USDC existing-market exit passed in a clean control and failed afterward. The stock mock does not implement fresh-child registration.

```
Ocelot-security-tests-4ef4ee4/integration/stock/child-first-claim.test.ts:27-42, 66-74

      trader_pda_index: c.trader_pda_index,
      parent_subaccount_index: c.parent_subaccount_index,
      trader_subaccount_index: c.trader_subaccount_index,
    }, [], { child_assignment: assignment });
    const { logs } = t.send(ix);
    assert.ok(t.account(assignment), 'assignment PDA must have been created');
    const stored = t.decode('PhoenixChildAssignment', assignment);
    assert.ok(stored.market.equals(t.market));
    assert.ok(stored.trader.equals(t.trader));
    assert.ok(!logs.some((line: string) => line.includes('venue-mock:')),
      'claim does not depend on an actual Phoenix child CPI');
    assert.deepEqual(t.counters(), { order: 0n, withdraw: 0n, transfer: 0n, deposit: 0n });

    // The same market may repeat its own configuration: a negative control for first-claim
        semantics.
    t.send(ix);
    assert.ok(t.decode('PhoenixChildAssignment', assignment).market.equals(t.market));
// ...
      parent_trader: c.parent_trader,
      trader: c.trader,
      authority_shard_id: c.authority_shard_id,
      trader_pda_index: c.trader_pda_index,
      parent_subaccount_index: c.parent_subaccount_index,
      trader_subaccount_index: c.trader_subaccount_index,
    }, [], { market: otherMarket, config: otherConfig, child_assignment: assignment });
    t.send(secondClaim, 'PhoenixAccounts');
    assert.equal(t.account(otherConfig), null, 'failed second claim must roll back config
        creation');
```

2.4 Rounding can weaken the shared order price bound

LOW SOLVED

Direct source tests show the low-price guard violation. Actual Phoenix matching at that price was not demonstrated.

The maintenance price check rounds both the Phoenix tick price and the allowed threshold into micro-USDC. At small prices, the integer floor can admit a materially worse limit than the stated ten-percent bound. The guard is shared by V1 mint and redemption callers as well as legacy paths.

2.4.1 Code location

packages/leveraged_vault/programs/leveraged_vault/src/lib.rs:1502–1511, 1517–1543

```
    } else {
        let scale = 10u128
            .checked_pow((-base_lot_decimals) as u32)
            .ok_or(LtError::Overflow)?;
        (price_ticks as u128)
            .checked_mul(tick_size as u128)
            .ok_or(LtError::Overflow)?
            .checked_div(scale)
            .ok_or(LtError::Overflow)?
    };
// ...
fn phoenix_validate_maintenance_price(
    place_market_order_args: &[u8],
    mark_price: u64,
    cfg: &PhoenixConfig,
) -> Result<()> {
    require!(mark_price > 0, LtError::OracleBad);
    let side = phoenix_ioc_side(place_market_order_args)?;
    let price = phoenix_price_micro_usd_from_ticks(
        phoenix_ioc_price_ticks(place_market_order_args)?,
        cfg.tick_size,
        cfg.base_lot_decimals,
    )?;
    if side == 0 {
        let ceiling = (mark_price as u128)
            .checked_mul((10_000 + PHOENIX_MAINTENANCE_MAX_SLIPPAGE_BPS) as u128)
            .ok_or(LtError::Overflow)?
            .checked_add(9_999)
            .ok_or(LtError::Overflow)?
            / 10_000;
        require!((price as u128) <= ceiling, LtError::PriceMoveTooLarge);
    } else if side == 1 {
        let floor = (mark_price as u128)
            .checked_mul((10_000 - PHOENIX_MAINTENANCE_MAX_SLIPPAGE_BPS) as u128)
            .ok_or(LtError::Overflow)?
            / 10_000;
        require!((price as u128) >= floor, LtError::PriceMoveTooLarge);
    } else {
```

2.4.2 Impact

The original guard accepted a one-micro-USDC sell limit at a two-micro mark, a 50% discount, although a 90-of-100 control passed and 89-of-100 failed. A negative-decimal case hid a 27.3% buy move. An affected real order could execute beyond the intended bound; live tick settings, matching and actual loss remain unverified.

2.4.3 Recommendation

To mitigate this issue, the following improvements are recommended: compare the native tick value and mark in one exact rational unit with checked cross-products, and enforce the intended execution bound after the order where the venue exposes fill data. Cover low-price and negative-decimal limits in regression tests.

2.4.4 Remediation status

SOLVED: Client fixed this issue in commit 8090fbe7 by checking native ticks against the exact slippage ratio before display rounding.

2.4.5 Proof of concept

Two direct Rust tests of the assessed guard passed. Earlier compiled tests exercised an off-mark V1 branch using a modified venue mock; the stock mock did not supply that adverse fill. These tests establish guard behavior, not real Phoenix execution at the edge prices.

Ocelot-security-tests-4ef4ee4/rust/modules/price_guard_checks.rs:26-44

```
    #[test]
    fn v1_redeem_shared_price_guard_allows_half_price_at_two_micro_mark() {
        let cfg = config(1, 0);
        assert!(phoenix_validate_maintenance_price(&order(1, 1), 2, &cfg).is_ok());
        assert!(phoenix_validate_redeem_order_args(&cfg, 1, 1, 2, &order(1, 1)).is_ok());
        assert!(phoenix_validate_maintenance_price(&order(1, 90), 100, &cfg).is_ok());
        assert!(phoenix_validate_maintenance_price(&order(1, 89), 100, &cfg).is_err());
    }

    #[test]
    fn negative_decimal_conversion_hides_more_than_ten_percent_price_move() {
        let cfg = config(1, -6);
        let observed_ticks = 1_100_000_u64;
        let offered_ticks = 1_400_000_u64;
        let observed = phoenix_price_micro_usd_from_ticks(observed_ticks, 1, -6).unwrap();
        let offered = phoenix_price_micro_usd_from_ticks(offered_ticks, 1, -6).unwrap();
        assert_eq!((observed, offered), (1, 1));
        assert!(phoenix_validate_maintenance_price(&order(0, offered_ticks), observed,
            &cfg).is_ok());
        assert!(offered_ticks as u128 * 10_000 > observed_ticks as u128 * 11_000);
```

2.5 A stressed partial redemption can fail the post-exit margin check

LOW SOLVED

The baseline stressed partial exit was reproduced locally. In PR #25, V3 returns a retirement error and V4 passes the tested risk-reducing exit.

A holder's partial exit can reduce exposure proportionally yet leave the remaining venue position above its absolute leverage cap. Ocelot checks that cap after the partial redemption and rolls back the exit. The full-close path is exempt, and an at-mark emergency reduction can restore the partial-exit path.

2.5.1 Code location

packages/leveraged_vault/programs/leveraged_vault/src/flows.rs:1115–1117

```
    if let Some(f) = &finance {
        f.venue_margin(m, cfg, &ctx.accounts.trader, mark_price, current_funding)?;
    }
```

packages/leveraged_vault/programs/leveraged_vault/src/finance_view.rs:219–230

```
pub fn require_venue_margin(snapshot: FinanceSnapshot, policy: FinancePolicy) -> Result<()> {
    if snapshot.base == 0 {
        return Ok(());
    }
    require!(
        snapshot.venue >= policy.min_venue_equity_usdc
            && snapshot.venue > 0
            && snapshot.notional as u128 * 10_000
                <= snapshot.venue as u128 * policy.max_venue_leverage_bps as u128,
        LtError::FinanceExecutionBound
    );
    Ok(())
```

2.5.2 Impact

The local effect is a recoverable delay to a holder's partial withdrawal during stress. It does not show permanent lock or theft. Whether a direct partial exit should be permitted when it does not worsen leverage is a product-policy point to confirm.

2.5.3 Recommendation

To mitigate this issue, the following improvements are recommended: define a stress-exit rule that permits risk-reducing partial withdrawals without weakening the venue safety limit, or provide a reliable reduction-and-exit path. Test stressed marks, cap boundaries, rollback and recovery.

2.5.4 Remediation status

SOLVED: Client fixed this issue in commit cd7aa827 by retiring V3 redemption; V4 supports the tested risk-reducing partial exits.

2.5.5 Proof of concept

Compiled stock-mock tests rejected and rolled back a 10% partial exit above a 3.5× cap. The same exit succeeded with a higher cap; a sole holder fully closed. A 10% mark fall produced a 3.86× post-exit state that failed at 3.5× and passed at 4×. A public at-mark emergency reduction restored the exit.

Ocelot-security-tests-4ef4ee4/integration/stock/stressed-partial-redeem.test.ts:20-39

```
  it('rejects and rolls back a valid 10 percent partial exit above the venue cap', async () => {
    const t = await configured(25_000, true);
    const before = t.amount(t.userUsdc);
    const result = t.send(t.redeemIx(10n * MICRO), 'FinanceExecutionBound');
    assert.equal(t.amount(t.userUsdc), before);
    assert.ok(result.logs.some((x) => x.includes('FinanceExecutionBound')));
  });

  it('allows the same partial exit at a cap above the initial 3x venue leverage', async () => {
    const t = await configured(35_000, true);
    t.send(t.redeemIx(10n * MICRO));
    assert.ok(t.amount(t.userUsdc) > 1_000n * MICRO - 100n * MICRO);
    assert.equal(t.supply(), 90n * MICRO);
  });

  it('allows a sole holder to close the entire out-of-policy position', async () => {
    const t = await configured(25_000, false);
    t.send(t.redeemIx(100n * MICRO));
    assert.equal(t.supply(), 0n);
  });
```

2.6 Authority transfer leaves the outgoing key with keeper access

LOW SOLVED

Ocelot confirmed the intended shared-role handoff. The baseline transition violates it; the proposed source change passed local replay.

A new market initially assigns its authority as keeper. When that authority nominates a successor and the successor accepts, the contract replaces authority but leaves keeper unchanged. If the roles were held by the same outgoing key, that key keeps keeper access after the transfer. Nikhil confirmed that both roles should move together in this case; an independently appointed keeper should remain in place.

2.6.1 Code location

packages/leveraged_vault/programs/leveraged_vault/src/lib.rs:216–220, 1218–1223

```
        let m = &mut ctx.accounts.market;
        m.authority = ctx.accounts.authority.key();
        m.pending_authority = Pubkey::default();
        m.keeper = ctx.accounts.authority.key(); // keeper defaults to authority; settable later
        m.usdc_mint = ctx.accounts.usdc_mint.key();
// ...
fn accept_market_authority_transfer(market: &mut Market, signer: Pubkey) -> Result<()> {
    require_keys_eq!(market.pending_authority, signer, LtError::AuthorityRequired);
    market.authority = signer;
    market.pending_authority = Pubkey::default();
    Ok(())
}
```

2.6.2 Impact

A former authority with its old signing key can still invoke keeper-gated maintenance, cash movement between the vault and venue, abandoned-share recovery, mark checkpoints and fee settlement until the new authority rotates keeper. These paths retain their own policy, custody and execution checks. The local proof establishes an unintended privilege that survives a handoff, not direct theft or a current deployed exposure. The new authority can remove it with set_keeper.

2.6.3 Recommendation

To mitigate this issue, the following improvements are recommended: on successful acceptance, if keeper still equals the outgoing authority, assign keeper to the new authority before replacing authority. Preserve separately appointed and disabled keepers. Add regression checks for shared, separate and failed transfers, and review completed handoffs for an outgoing keeper that should be rotated. PR #10 proposes this behavior but remains subject to full integration and deployment verification.

2.6.4 Remediation status

SOLVED: Client fixed this issue in commit a9e7021e by transferring the keeper role with authority when the outgoing key held both roles.

2.6.5 Proof of concept

In an isolated copy of the assessed source, a new shared-role handoff test failed on the baseline: acceptance changed authority but left keeper equal to the outgoing key. Independently appointed keeper and wrong-signer controls passed. Replaying PR #10's exact conditional runtime edit made all three tests pass; the 117-test host Rust suite also passed. Six symbolic transition checks supported the same result. The complete stacked PR, real account state and deployed executable were not independently tested.

```
Ocelot-security-tests-4ef4ee4/rust/modules/authority_interleavings.rs:14-37, 40-51

#[test]
fn default_keeper_survives_authority_handoff() {
    let old = Pubkey::new_unique();
    let new = Pubkey::new_unique();
    let outsider = Pubkey::new_unique();
    let mut market = market_with_roles(old, old);

    initiate_market_authority_transfer(&mut market, new).unwrap();
    assert!(accept_market_authority_transfer(&mut market, outsider).is_err());
    assert_eq!(market.authority, old);
    assert_eq!(market.keeper, old);

    accept_market_authority_transfer(&mut market, new).unwrap();
    assert_eq!(market.authority, new);
    assert_eq!(market.pending_authority, Pubkey::default());
    assert_eq!(market.keeper, old);
    assert_ne!(old, market.authority); // AuthorityOnly / fee payout fails for old.
    assert!(old == market.keeper || old == market.authority); // FinanceView::require_keeper
        accepts old.
    assert!(new == market.keeper || new == market.authority);
    assert!(!(outsider == market.keeper || outsider == market.authority));

    // Authorized set_keeper is the separate revocation step.
    market.keeper = new;
    assert!(!(old == market.keeper || old == market.authority));
// ...
#[test]
fn dedicated_keeper_is_intentionally_separate_from_authority() {
    let old = Pubkey::new_unique();
    let new = Pubkey::new_unique();
    let dedicated = Pubkey::new_unique();
    let mut market = market_with_roles(old, dedicated);
    initiate_market_authority_transfer(&mut market, new).unwrap();
    accept_market_authority_transfer(&mut market, new).unwrap();
    assert_eq!(market.keeper, dedicated);
    assert_ne!(old, market.keeper);
    assert!(!(old == market.keeper || old == market.authority));
    assert!(dedicated == market.keeper || dedicated == market.authority);
```

2.7 Legacy emergency de-risking can accept an adverse execution

INFORMATIONAL SOLVED

The reviewed source blocks this legacy route before order execution.

On the V0 path, any caller can trigger emergency de-risking after the danger condition. The final check requires positive equity but has no V1-style post-trade loss budget. A materially adverse fill can therefore be accepted while holders absorb the loss. This requires a funded, callable V0 market and the corresponding real venue fill.

2.7.1 Code location

packages/leveraged_vault/programs/leveraged_vault/src/lib.rs:1090–1097

```
            )?
        };
        // A force-close is best-effort: exhausting gross equity reverts the
        // entire transaction, leaving the position open. This equity value floors
        // deficits to zero, so accepting zero here would also accept negative equity.
        require!(after_equity > 0, LtError::PoolWiped);
        m.last_price = mark_price;
        m.nav = m.marked_nav(after_equity)?;
```

2.7.2 Impact

The modified-venue local case reduced modeled holder equity from 60 to 48 USDC at a 10% adverse fill. An at-mark V0 control stayed at 60 USDC, and comparable V1 execution rolled back. The 15 October inventory listed all 26 recorded markets as V1; no current funded V0 market or real hostile fill was established.

2.7.3 Recommendation

To mitigate this issue, the following improvements are recommended: apply a bounded post-trade loss or execution floor to legacy emergency reduction. If no funded V0 market remains, remove or make the legacy route unreachable.

2.7.4 Remediation status

SOLVED: Client fixed this issue in commit 74e4024 by retiring the legacy de-risk instruction before venue execution or accounting changes.

2.7.5 Proof of concept

The compiled Ocelot SBF was exercised against a test-only Phoenix mock with explicit off-mark execution. The stock mock executed at mark and did not reproduce the adverse cases. This is a conditional legacy source behavior.

Ocelot-security-tests-4ef4ee4/integration/adverse-fill/legacy-derisk-loss.test.ts:29-50

```
  it('outsider sells at the allowed ten-percent adverse limit and removes 12 USDC from V0
      holders', async () => {
    const t = await stressed(true);
    t.control({ mode: Mode.AdverseFill, triggers: Trigger.Order, executionLoss: 12n * MICRO });
    const { logs } = t.send(t.instruction('phoenix_de_risk', {
      place_market_order_args: closeOrder(900n),
    }, t.groups.order, { cranker: t.user.publicKey }));
    assert.equal(t.supply(), SUPPLY);
    assert.equal(t.events(logs, 'PhoenixMaintenanceSubmitted').length, 1);
    assert.equal(integer(t.decode('Market', t.market).nav), 480_000n);
    const trader = t.account(t.trader)!.data;
    assert.equal(trader.readBigInt64LE(248), 180n);
    assert.equal(trader.readBigInt64LE(88) + trader.readBigInt64LE(248) * MICRO +
        trader.readBigInt64LE(256), 48n * MICRO);
  });

  it('the same V0 reduction at the mark preserves all 60 USDC of holder equity', async () => {
    const t = await stressed(true);
    t.send(t.instruction('phoenix_de_risk', {
      place_market_order_args: closeOrder(1000n),
    }, t.groups.order, { cranker: t.user.publicKey }));
    assert.equal(integer(t.decode('Market', t.market).nav), 600_000n);
    const trader = t.account(t.trader)!.data;
    assert.equal(trader.readBigInt64LE(88) + trader.readBigInt64LE(248) * MICRO +
        trader.readBigInt64LE(256), 60n * MICRO);
```

2.8 Legacy minting can issue shares despite an adverse fill

INFORMATIONAL SOLVED

The reviewed source blocks the legacy share-issuance route before economic execution.

V0 credits the increase in collateral rather than the change in gross equity after its order. If execution loses value, the minter can receive shares based on more value than the vault gained, diluting existing holders. V1 uses the post-order equity and settlement calculation instead.

2.8.1 Code location

packages/leveraged_vault/programs/leveraged_vault/src/flows.rs:455–484

```
    if let Some(f) = &finance {
        after_equity = f.gross(m, cfg, &ctx.accounts.trader, mark_price, current_funding)?;
        f.venue_margin(m, cfg, &ctx.accounts.trader, mark_price, current_funding)?;
    }
    require!(after_base == expected_after_base, LtError::PhoenixAccounts);
    let added = if bounds.is_some() {
        // Liability extinguishment is value contributed even though it bypasses Phoenix.
        let value = (after_equity as u128)
            .checked_add(settlement as u128)
            .ok_or(LtError::Overflow)?
            .checked_sub(before_equity as u128)
            .ok_or(LtError::NothingToSettle)?;
        u64::try_from(value).map_err(|_| error!(LtError::Overflow))?
    } else {
        let value = after_collateral
            .checked_sub(before_collateral)
            .ok_or(LtError::NothingToSettle)?;
        u64::try_from(value).map_err(|_| error!(LtError::NothingToSettle))?
    };
    require!(added > 0, LtError::NothingToSettle);
    // V1 uses marked equity and may include favorable execution; legacy collateral credit is
        capped.
    if bounds.is_none() {
        require!(added <= deposit_amount, LtError::PhoenixAccounts);
    }
    let shares = if bounds.is_some() {
        streaming::mint_shares(added, net_equity_before, m.total_shares)?
    } else {
        venue::nav::mint_shares(added, m.nav)
    };
    streaming::require_mint_rounding_bound(added, net_equity_before, m.total_shares, shares)?;
```

2.8.2 Impact

A modified-venue local case imposed an 0.087 USDC fill loss and lowered the incumbent's 100 USDC claim to 99.920901 USDC. The matched V1 case issued fewer shares and preserved the incumbent claim. Current V0 reachability and real Phoenix adverse matching were not shown.

2.8.3 Recommendation

To mitigate this issue, the following improvements are recommended: base credited mint value on the net post-order equity change and settlement, as in V1, or retire the V0 mint route.

2.8.4 Remediation status

SOLVED: Client fixed this issue in commit 74e4024 by retiring legacy mint and redemption before collateral or shares can change.

2.8.5 Proof of concept

The compiled-program differential passed with an off-mark test venue; the stock mock filled at mark and failed the adverse-fill assertion. The dated market inventory lists V1, so the result is conditional on a callable funded V0 market.

Ocelot-security-tests-4ef4ee4/integration/adverse-fill/legacy-mint-loss.test.ts:43-59

```
  it('v0 credits all collateral despite adverse IOC execution, diluting pre-existing holders',
      async () => {
    const normal = await mint(true, 0n);
    const adverse = await mint(true, 87_000n);
    assert.equal(normal.grossAfter, 109_990_000n);
    assert.equal(normal.minted, 9_990_000n);
    assert.equal(normal.oldHolderClaim, 100n * MICRO);
    assert.equal(adverse.grossAfter, normal.grossAfter - 87_000n);
    assert.equal(adverse.minted, normal.minted);
    assert.ok(adverse.oldHolderClaim < normal.oldHolderClaim);
    assert.equal(adverse.oldHolderClaim, 99_920_901n);
  });

  it('v1 issues fewer shares for the same adverse IOC execution', async () => {
    const adverse = await mint(false, 87_000n);
    assert.equal(adverse.grossAfter, 109_903_000n);
    assert.equal(adverse.minted, 9_903_000n);
    assert.equal(adverse.oldHolderClaim, 100n * MICRO);
```

2.9 Legacy floored NAV can overissue shares in an extreme state

INFORMATIONAL SOLVED

The reviewed source blocks the legacy NAV route before economic execution.

V0 floors NAV per share to a whole micro-unit before dividing credited value by that NAV. At very high share supply and heavily depleted backing, the floor can make the issued-share count exceed an exact-ratio calculation. The mint rounding guard uses saturating subtraction, so it does not reject overissue on this side.

2.9.1 Code location

packages/leveraged_vault/programs/leveraged_vault/src/venue/nav.rs:8-25

```
pub fn nav_per_share(equity: u64, total_shares: u64) -> u64 {
    if total_shares == 0 {
        return SCALE as u64;
    }
    let n = (equity as u128 * SCALE / total_shares as u128) as u64;
    if n == 0 {
        1
    } else {
        n
    }
}

/// Shares to mint for `value_added` (micro-USD) at the current `nav` (1e6). Rounds down
    (pool-favorable).
pub fn mint_shares(value_added: u64, nav: u64) -> u64 {
    if nav == 0 {
        return value_added;
    }
    (value_added as u128 * SCALE / nav as u128) as u64
```

packages/leveraged_vault/programs/leveraged_vault/src/flows.rs:475-484

```
    // V1 uses marked equity and may include favorable execution; legacy collateral credit is
        capped.
    if bounds.is_none() {
        require!(added <= deposit_amount, LtError::PhoenixAccounts);
    }
    let shares = if bounds.is_some() {
        streaming::mint_shares(added, net_equity_before, m.total_shares)?
    } else {
        venue::nav::mint_shares(added, m.nav)
    };
    streaming::require_mint_rounding_bound(added, net_equity_before, m.total_shares, shares)?;
```

2.9.2 Impact

A synthetic 200-million-share, 380-USDC backing state issued 99,900,000,000,000 V0 shares for a 99.9-USDC credited mint, versus 52,578,947,368,421 by the V1 exact ratio. The model estimated 59.959986 USDC incumbent loss. Reaching that state through real market history, a complete handler execution at scale and any funded callable V0 market remain unproved.

2.9.3 Recommendation

To mitigate this issue, the following improvements are recommended: use exact-ratio share issuance in V0, compare overissue and underissue symmetrically, and gate the legacy path if it has been retired.

2.9.4 Remediation status

SOLVED: Client fixed this issue in commit 74e4024 by retiring legacy mark and rebalance routes before V0 NAV arithmetic can run.

2.9.5 Proof of concept

A source arithmetic harness and direct Rust test of the actual rounding guard reproduced the V0 result. Exact-ratio V1 and ordinary-NAV controls did not overissue. The fee-adjusted round-trip figure in the synthetic model is not a measured venue profit.

```
Ocelot-security-tests-4ef4ee4/integration/stock/legacy-rounded-nav.test.ts:24-34

  it('overissues against a fractional sub-micro NAV', async () => {
    const result = await caseWithGross(150n);
    assert.equal(result.minted, 9_990_000_000_000n);
    assert.equal(result.exactFair, 6_660_000_000_000n);
    assert.equal(result.incumbentClaim, 100n);
  });

  it('issues the exact ratio when the sub-micro NAV is divisible', async () => {
    const result = await caseWithGross(200n);
    assert.equal(result.minted, result.exactFair);
    assert.equal(result.incumbentClaim, 200n);
```

2.10 Post-order valuation can reuse the pre-order mark

INFORMATIONAL SOLVED

The reviewed source uses the observed post-order mark for pool valuation and risk checks without assigning its change to newly minted shares.

FinanceView checks that the funding index did not change during a Phoenix order, then values the resulting trader at the mark captured before that order. If the real venue updates PerpAssetMap's mark without changing funding in that call, mint share accounting and maintenance limits can use an outdated price. The supplied builder marks the map writable, but the real handler transition was not shown.

2.10.1 Code location

packages/leveraged_vault/programs/leveraged_vault/src/finance_view.rs:185–198

```
    pub fn after(&self, before: FinanceSnapshot) -> Result<FinanceSnapshot> {
        require!(
            read_phoenix_mark(&self.perp_asset_map, &self.config)?.1 == before.funding,
            LtError::OracleBad
        );
        cash_surplus(self.finance.accounted_cash, self.cash_vault.amount)?;
        snapshot_at(
            &self.market,
            &self.config,
            &self.trader,
            self.finance.accounted_cash,
            before.mark,
            before.funding,
        )
```

2.10.2 Impact

In local long-rising and short-falling mark cases, about 0.11 USDC of incumbent repricing was allocated to a new minter. A maintenance case passed at the old mark despite 4.0303× leverage at the current mark. These effects depend on a real same-CPI mark change without funding movement.

2.10.3 Recommendation

To mitigate this issue, the following improvements are recommended: re-read mark and funding after the order and value the resulting position consistently, or reject an in-call mark change before applying share and risk checks.

2.10.4 Remediation status

SOLVED: Client fixed this issue in commit 855c3cb by separating share credit from post-fill mark changes and using the observed mark for NAV and margin.

2.10.5 Proof of concept

Compiled tests made the stock mock change PerpAssetMap during PlaceMarketOrder and used unchanged-mark and threshold controls. They do not establish that deployed Phoenix makes this transition.

```
Ocelot-security-tests-4ef4ee4/integration/stock/intra-cpi-mark-mint.test.ts:74-81

    it(`${short ? 'short falling' : 'long rising'} mark transfers part of incumbent repricing to
        the new minter`, async () => {
      const neutral = await run(short, 0n);
      const moved = await run(short, movement);
      assert.ok(neutral.fairNewClaim - neutral.newClaim <= 1n);
      assert.ok(neutral.oldClaim - neutral.fairOldClaim <= 1n);
      assert.ok(moved.newClaim > moved.fairNewClaim);
      assert.ok(moved.oldClaim < moved.fairOldClaim);
      assert.ok(moved.newClaim - moved.fairNewClaim > 100_000n);
```

2.11 High bits in a Phoenix position key can block marking

INFORMATIONAL SOLVED

The reviewed decoder accepts the supported high-bit storage representation while preserving the configured logical asset identity.

Ocelot compares the complete u64 position key with the configured asset ID. The copied Phoenix decoder derives the position asset from the low 32 bits. If a legitimate trader position uses nonzero high bits, Ocelot can miss its configured position and reject finance marking.

2.11.1 Code location

packages/leveraged_vault/programs/leveraged_vault/src/venue/phoenix.rs:495–515

```
    pub fn read_position(data: &[u8], asset_id: u64) -> Option<(i64, i64, i64, i64)> {
        if data.len() < OFF_ENTRIES {
            return None;
        }
        let collateral = i64_at(data, OFF_COLLATERAL);
        let len = u64_at(data, OFF_POSITIONS_LEN) as usize;
        for i in 0..len {
            let e = OFF_ENTRIES + i * ENTRY_BYTES;
            if data.len() < e + ENTRY_BYTES {
                return None;
            }
            if u64_at(data, e) == asset_id {
                return Some((
                    collateral,
                    i64_at(data, e + E_BASE),
                    i64_at(data, e + E_VQUOTE),
                    i64_at(data, e + E_CUMFUND),
                ));
            }
        }
        None
```

2.11.2 Impact

The local effect is an unavailable finance path for a synthetic Phoenix-owned trader with such a key. No actual Phoenix writer, current trader account or funded affected market with nonzero high bits has been shown.

2.11.3 Recommendation

To mitigate this issue, the following improvements are recommended: decode the key with the same asset-ID rule as Phoenix and test high-bit and ordinary keys against real writer-generated account layouts.

2.11.4 Remediation status

SOLVED: Client fixed this issue in commit 855c3cb by decoding the Phoenix child map by asset identity and accepting supported storage discriminants.

2.11.5 Proof of concept

A direct Rust parser test passed. In a compiled stock-mock case, key (1<<32)|1 returned PhoenixAccounts; the same low-asset key 1 passed. The account bytes were synthetic fixture data.

```
Ocelot-security-tests-4ef4ee4/integration/stock/position-map-challenges.test.ts:21-27

  it('rejects a position key with nonzero high bits while accepting the same low asset key', async
      () => {
    const t = await fixture({ liability: 0n });
    editKey(t, (1n << 32n) | 1n);
    mark(t, 'PhoenixAccounts');
    editKey(t, 1n);
    assert.ok(mark(t).logs.some((x) => x.includes('success')));
  });
```

2.12 A non-position-only trader map can block finance marking

INFORMATIONAL SOLVED

The reviewed parser recognizes unrelated collateral in a flat child, and the cleanup and mint flow clears it atomically in local real-venue tests.

When Ocelot cannot find its configured perp position, it treats a trader as flat only if the raw map length is zero. The copied SDK distinguishes non-position entries from perp positions. A map containing only such an entry can therefore be rejected even though it has no relevant position.

2.12.1 Code location

packages/leveraged_vault/programs/leveraged_vault/src/venue/phoenix.rs:488–506

```
    pub fn has_no_positions(data: &[u8]) -> bool {
        data.len() >= OFF_ENTRIES && u64_at(data, OFF_POSITIONS_LEN) == 0
    }

    /// `(quote_lot_collateral, base_lot_position, virtual_quote_lot_position,
        cumulative_funding_snapshot)`
    /// for `asset_id`, or `None` if the account is too short or has no such position. All in
        their raw
    /// i64 units (quote lots = micro-USD; base lots; cumulative funding snapshot).
    pub fn read_position(data: &[u8], asset_id: u64) -> Option<(i64, i64, i64, i64)> {
        if data.len() < OFF_ENTRIES {
            return None;
        }
        let collateral = i64_at(data, OFF_COLLATERAL);
        let len = u64_at(data, OFF_POSITIONS_LEN) as usize;
        for i in 0..len {
            let e = OFF_ENTRIES + i * ENTRY_BYTES;
            if data.len() < e + ENTRY_BYTES {
                return None;
            }
            if u64_at(data, e) == asset_id {
```

2.12.2 Impact

At the original audit cutoff, the synthetic collateral entry blocked finance marking. Follow-up Phoenix tests confirmed that an unrelated actor can create a corresponding entry in an empty child. The reviewed parser now treats that child as flat, and cleanup supports reopening.

2.12.3 Recommendation

To mitigate this issue, the following improvements are recommended: classify map entries by their Phoenix type and configured asset rather than raw length, then test writer-generated non-position entries and empty maps.

2.12.4 Remediation status

SOLVED: Client fixed this issue in commits e632fae and aa1876d by identifying unrelated native and SPL collateral and adding atomic cleanup before minting.

2.12.5 Proof of concept

The original parser and compiled tests reproduced the rejection with synthetic trader bytes. Follow-up direct Phoenix tests created donated native and SPL entries; full local Ocelot, Phoenix and Ember transactions passed atomic cleanup and mint in the PR #23 candidate.

```
Ocelot-security-tests-4ef4ee4/integration/stock/position-map-challenges.test.ts:29-35

  it('rejects a non-position-only map while accepting a truly empty map', async () => {
    const t = await fixture({ liability: 0n });
    editKey(t, 0xFFFF_0000n);
    mark(t, 'PhoenixAccounts');
    editKey(t, 0xFFFF_0000n, 0n);
    mark(t);
  });
```

2.13 Accrued fee allocation can be impaired after an adverse redemption

INFORMATIONAL SOLVED

The reviewed source rejects adverse user exits that would impair the allocated streaming fee.

A redemption can realize less proceeds than the streaming-fee liability allocated to the exiting shares. The split retires the unpaid amount. This is an accounting and fee-priority effect; the current proof does not show independent attacker profit or an incorrect holder payout.

2.13.1 Code location

packages/leveraged_vault/programs/leveraged_vault/src/streaming.rs:182–195

```
pub fn redeem_split(
    proceeds: u64,
    allocated_liability: u64,
    redemption_bps: u16,
) -> Result<(u64, u64, u64, u64)> {
    let paid = allocated_liability.min(proceeds);
    let impaired = allocated_liability - paid;
    let holder_proceeds = proceeds - paid;
    let fee = bps_fee(holder_proceeds, redemption_bps)?;
    Ok((
        paid,
        impaired,
        fee,
        holder_proceeds.checked_sub(fee).ok_or(LtError::Overflow)?,
```

packages/leveraged_vault/programs/leveraged_vault/src/flows.rs:1052–1083

```
    let (streaming_paid, impaired, fee, net) = streaming::redeem_split(
        proceeds,
        if keeper_settlement {
            proceeds
        } else {
            allocated_fee
        },
        m.redemption_fee_bps,
    )?;
    if let Some(b) = &bounds {
        require!(
            (if keeper_settlement {
                streaming_paid
            } else {
                net
            }) >= b.min_usdc,
            LtError::MinimumOutput
        );
    }
    let vault_transfer = fee.checked_add(streaming_paid).ok_or(LtError::Overflow)?;
    let retired_remainder = if full_close {
        m.streaming_fee_remainder
    } else {
        0
    };
    if bounds.is_some() {
```

packages/leveraged_vault/programs/leveraged_vault/src/flows.rs:1052–1083 (continued)

```
        streaming::settle(
            m,
            streaming_paid,
            impaired,
            if keeper_settlement { 2 } else { 1 },
        )?;
```

2.13.2 Impact

In a modified-venue case after one year of accrual, 0.05 USDC of a 0.10-USDC allocated fee was impaired, while the exiting holder received zero. At-mark and shorter-accrual controls paid the fee. Real maker-side capture, venue cost and net attacker gain were not demonstrated.

2.13.3 Recommendation

To mitigate this issue, the following improvements are recommended: confirm the intended priority of accrued fees under loss, and, if the liability must be protected, separate its backing or enforce a redemption settlement rule that preserves it.

2.13.4 Remediation status

SOLVED: Client fixed this issue in commits c625206 and 855c3cb by bounding redemption price and loss while preserving the allocated streaming fee.

2.13.5 Proof of concept

The adverse case required an off-mark test venue; the stock mock rejected its adverse-fill assertion. The result establishes conditional fee-priority behavior in local accounting, not an economic exploit.

Ocelot-security-tests-4ef4ee4/integration/adverse-fill/fee-impairment.test.ts:181–185, 206–216

```
  it('impairs a one-percent fee liability at 1.99x venue leverage and a two-micro mark', async ()
      => {
    const t = await fixture({ liability: MICRO });
    t.token(t.userLt, t.ltMint, t.user.publicKey, 10n * MICRO);
    t.token(t.keeperLt, t.ltMint, t.keeper.publicKey, 90n * MICRO);
    await t.replace('PhoenixConfig', t.config, { tick_size: bn(1) });
// ...
      ember_withdraw_account_count: t.groups.ember.length,
      min_usdc: bn(0), deadline: bn(NOW + 60n), expected_revision: bn(1),
    }, [...t.groups.order, ...t.groups.withdrawal, ...t.groups.ember]));
    const event = t.events(logs, 'StreamingFeeSettledV1')[0];
    assert.equal(integer(event.paid_usdc), 50_000n);
    assert.equal(integer(event.impaired_usdc), 50_000n);
    assert.equal(integer(t.decode('Market', t.market).reserved_u64_0), 900_000n);
    const after = t.account(t.trader)!.data;
    const remainingGross = after.readBigInt64LE(88) + after.readBigInt64LE(248) * 2n +
        after.readBigInt64LE(256);
    assert.equal(remainingGross, 90n * MICRO);
    assert.equal(remainingGross - integer(t.decode('Market', t.market).reserved_u64_0),
        89_100_000n);
```

IRONNODE

ironnode.io

OCELOT / FINAL REPORT

Thank you for choosing IronNode.

PREPARED FOR OCELOT / 17 NOVEMBER 2026