Accretion

SECURITY ASSESSMENT

Provided by Accretion Labs Pte Ltd. for Manifest
May 05, 2026
A26MAN1

Manifest

AUDITORS

Role	Name
Lead Auditor	Robert Reith (robert@accretion.xyz)
Senior Auditor	Mahdi Rostami (mahdi@accretion.xyz)
Auditor	Niklas Brymko (niklas@accretion.xyz)

CLIENT

Manifest (https://manifest.trade) engaged Accretion to conduct a security assessment of the Destiny vaults protocol, a decentralized asset management system on Solana that enables vault creation and multi-asset trading operations

ENGAGEMENT TIMELINE

20 Apr — Project Kickoff
Initial planning and scope definition

20 Apr — Assessment Begins
Security review and testing phase

27 Apr — Review Fixes
Security recommendations are given and implemented

27 Apr — Project Completion
Report delivery and on-chain confirmation

AUDITED CODE

#	Program ID	Repository
1	FATEX7qHcbWRip7scfUMFa1VbLWZzscy8Bfj68LexYtP	https://github.com/Bonasa-Tech/destiny
2	DSTNyVLdoKxTYSjzCQ5191wozVChdVD8qCcFWZwMMX37	https://github.com/Bonasa-Tech/destiny
3	FoRTUqpCaR3MYs2VFYuQNkzfJruzDWxxiJEGk9bFhib7	https://github.com/Bonasa-Tech/destiny

ASSESSMENT

The security assessment of Manifest's Destiny revealed 24 issues across the codebase. The severity breakdown includes 0 critical, 0 high, 10 medium, 7 low, and 7 informational issues. The development team demonstrated strong remediation practices, successfully fixing 21 of the 25 identified issues. 3 issues were acknowledged but not fixed due to risk acceptance or design trade-offs.

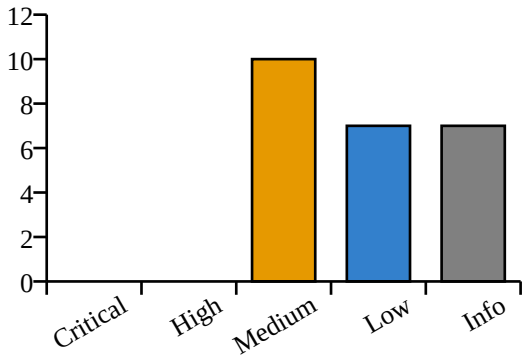
CODE ASSESSMENT

The codebase demonstrates solid Solana program architecture with proper PDA usage and CPI implementations. Security practices include appropriate account validations and access controls, though some edge cases in reward distribution and token handling were identified.

KEY FINDINGS

No critical or high-severity vulnerabilities were identified. All findings were medium severity or below, primarily involving edge cases in reward distribution logic, token handling, and access control validations that could lead to DoS or accounting issues.

SEVERITY DISTRIBUTION



ENGAGEMENT SCOPE

The scope of this security assessment was a full review of the following items:

Item 1: Fate
Link: https://github.com/Bonasa-Tech/destiny
Commit: 9c541d0f7a06fa71f7c19b59a0b7bb649a38d901
Program ID: FATEX7qHcbWRip7scfUMFa1VbLWZzscy8Bfj68LexYtP
Audit Result:
• **Audited Commit:** eafd30280b68c37ea8a055590a46cf41810366e5
• **Build Hash:** 0d4d4515d112252ea580ca680563521f620f0683af185950aaf06ba34d2a44bc
• **Status:** unverified
• **Comment:** On-chain version differs

Item 2: Destiny
Link: https://github.com/Bonasa-Tech/destiny
Commit: 9c541d0f7a06fa71f7c19b59a0b7bb649a38d901
Program ID: DSTNyVLdoKxTYSjzCQ5191wozVChdVD8qCcFWZwMMX37
Audit Result:
• **Audited Commit:** eafd30280b68c37ea8a055590a46cf41810366e5
• **Build Hash:** 25bbed1c797529687c1720ee8f6d4e8ede807e0391d47b9dcbf58e1bca8b8515
• **Status:** unverified
• **Comment:** On-chain version differs.

Item 3: Fortune
Link: https://github.com/Bonasa-Tech/destiny
Commit: 9c541d0f7a06fa71f7c19b59a0b7bb649a38d901
Program ID: FoRTUqpCaR3MYs2VFYuQNkzfJruzDWxxiJEGk9bFhib7
Audit Result:
• **Audited Commit:** eafd30280b68c37ea8a055590a46cf41810366e5
• **Build Hash:** 1b210f32352900ad072850e5899aa3177bb0ef31f997fbd24e511f5f6cd3eb03
• **Status:** unverified
• **Comment:** On-chain version differs

ISSUES SUMMARY

ID	TITLE	SEVERITY	STATUS
ACC-M1	Destiny `mint_lp` takes asset fair values directly from trader-supplied instruction data	MEDIUM	ACKNOWLEDGED
ACC-M2	LP global seat-table can be grief-filled, permanently bricking the vault's LP distribution path	MEDIUM	FIXED
ACC-M3	Token program substitution allows timer reset without distribution	MEDIUM	FIXED
ACC-M4	Free Eviction Enables Rent Griefing	MEDIUM	FIXED
ACC-M5	Wrong PDA Seeds Break Global Account Recovery Path	MEDIUM	FIXED
ACC-M6	AUM Desync Between Fortune and Destiny Can Over-Distribute Rewards	MEDIUM	FIXED
ACC-M7	Hardcoded 6-Decimal Assumption Can Break Reward Accounting	MEDIUM	FIXED
ACC-M8	Zero start_of_last_mint_float Can DoS Reward Distribution	MEDIUM	FIXED
ACC-M9	Fortune Rewards State Could Be Permanently Captured by a Frontrunner	MEDIUM	FIXED
ACC-M10	Off-Chain Signers Can Be Forwarded as CPI Signers	MEDIUM	FIXED
ACC-L1	`manifest_batch_update` forwards `instruction_data` verbatim with no discriminator pin	LOW	FIXED
ACC-L2	Destiny `mint_lp` transfers profit-share LP to an unvalidated `profit_share_token`	LOW	FIXED
ACC-L3	Destiny `set_owner` / `set_trader` have no two-step handover	LOW	ACKNOWLEDGED
ACC-L4	mint_lp Does Not Reduce Vault LP Balance After Sending Trader Share	LOW	FIXED
ACC-L5	Token-2022 Transfer Fees Cause Reward Accounting Mismatch	LOW	FIXED
ACC-L6	Delayed Reward Processing Can Underpay Rewards	LOW	ACKNOWLEDGED
ACC-L7	Reward Distributor Missing Vault Token Account Validation	LOW	FIXED
ACC-I1	Unguarded debug `msg!` / `log!` spam across hot paths	INFO	FIXED
ACC-I2	`VaultState.state_bump` is actually the vault PDA bump (naming drift)	INFO	FIXED
ACC-I3	Fate could be called as CPI and signature assembled	INFO	FIXED
ACC-I4	Wrong Offset Used for Remaining Accounts in split_manifest_market_account_infos	INFO	FIXED

ACC-I5	APY vs APR Mislabeling in target_apy_bps	INFO	FIXED
ACC-I6	Wrong Rewards Mint Could Be Initialized and Break Reward Distribution	INFO	FIXED
ACC-I7	Fortune Initialize Relies on Side Effect Instead of Rejecting Reinitialization	INFO	FIXED

DETAILED ISSUES

ACC-M1 Destiny `mint_lp` takes asset fair values directly from trader-supplied instruction data

MEDIUM ACKNOWLEDGED

Description

For every traded asset, `process_mint_lp` reads `(fair_value_mantissa, exponent)` out of the instruction data and writes it straight into `balance_cache.fair_value`. The resulting AUM feeds the HWM / profit-share calculation in `settle_lp_price`. A trader who either inflates prices above fair or who continuously lies by small increments can:

1. Force the LP price across the HWM and trigger profit-share issuance (L-05 is the free choice of recipient). 2. Mint more LP at an inflated perceived price, diluting external holders on every `mint_lp` cycle. 3. Corrupt the LP fair value that Fortune subsequently reads via `last_epoch_aum` / `last_epoch_float`.

The protocol's design assumes `trader` is honest and well-behaved (the README describes a single off-chain risk-manager key). The on-chain program has no oracle, no staleness check, and no sanity bounds: a single bad quote is accepted.

Location

https://github.com/Bonasa-Tech/destiny/blob/9c541d0f7a06fa71f7c19b59a0b7bb649a38d901/programs/destiny/src/processor/mint_lp.rs#L142-L167

Relevant Code

```
/// programs/destiny/src/processor/mint_lp.rs L142-L167
// Set fair value only for assets that are different from quote and lp.
let num_traded_assets = vault_state.balance_caches.len() - 2;

// Parse instruction data
let (header, instruction_data) = MintLPInstructionHeader::split_header(instruction_data)?;
let (asset_configs, _) =
    MintLPInstructionHeader::split_traded_asset_config(num_traded_assets, instruction_data)?;

msg!("recalculate AUM");
vault_state.last_epoch_aum = {
    // Set fair values for all traded assets and calculate portfolio value
    let mut portfolio_value = QuoteAtoms::new(vault_state.quote_balance().balance.as_u64());
    for (cache, config) in vault_state
        .balance_caches
        .iter_mut()
        .skip(2)
        .zip(asset_configs)
    {
        cache.set_fair_value(config.fair_value_mantissa, config.exponent)?;
        portfolio_value += cache
            .fair_value
            .checked_quote_for_base(cache.balance, false)
            .map_err(|_| ProgramError::ArithmeticOverflow)?;
    }
    portfolio_value
};
```

Mitigation Suggestion

In order of decreasing strength: 1. Sign every `mint_lp` price push with an oracle (Pyth / Switchboard) and reject trader-supplied prices outright. 2. Require `fate`-based co-signing so at least two keys must agree on the quote before it commits. 3. Enforce a per-epoch bound (e.g. AUM change ≤ N bps relative to previous AUM) so a single compromise has a bounded blast radius.

Remediation

Acknowledged. In practice, fate-based cosigning with co-signer side price oracle checks will be implemented, however, a hard requirement for fate cosigning will not be implemented because during configuration and setup a single signer setup will be used temporarily.

ACC-M2 LP global seat-table can be grief-filled, permanently bricking the vault's LP distribution path

MEDIUM FIXED

Description

`destiny::CreateVault` calls manifest's `GlobalCreate` for the freshly minted LP token, leaving an empty global account whose seat table is open to anyone. Manifest's `GlobalAddTrader` (programs/manifest/src/program/processor/global_add_trader.rs L17-L58) has no allow-list: any signer that pays the stranded seat fee plus the realloc rent can claim a seat under their own pubkey, capped at `MAX_GLOBAL_SEATS = 999` per global (programs/manifest/src/state/constants.rs L49).

A griefer who pre-claims all 999 seats in the LP global before destiny's `authority` PDA gets its first sweep in (the very first `mint_lp` for that vault) traps `authority` outside the seat table. From that point on, every `sweep_token_to_global` call in `mint_lp` hits the early-return branch in `programs/destiny/src/manifest.rs:104-110`:

```
if !global.has_global_seat(&(*authority.key()).into()) {
    if global.fixed.needs_eviction() {
        msg!("eviction not implemeted yet. skipping sweep to global");
        return Ok(());
    } else {
        true
    }
}
```

The CPI silently succeeds (`Ok(())`) but no LP tokens move. Freshly minted LP stays parked in `authority`'s LP ATA forever, and manifest's global-deposit rail (the only path destiny uses to distribute LP tokens via the order book) is permanently blocked for that vault.

Manifest's intended escape hatch is `GlobalEvict` (programs/manifest/src/program/processor/global_evict.rs): anyone can displace the seat with the smallest deposit balance by depositing strictly more. Since the griefer's seats hold zero LP atoms, displacing them is trivially cheap (1 atom of LP suffices). However, destiny does not wrap `GlobalEvict`: see `programs/destiny/src/manifest.rs:94-228`, which only exposes `GlobalAddTrader`, `GlobalDeposit`, `GlobalWithdraw`, and `GlobalCreate`. There is no on-chain code path through which `authority` (a destiny-owned PDA) can evict an existing seat. Recovery requires either a destiny program upgrade that adds a `GlobalEvict` wrapper, or atomic bundling of `CreateVault` with the first sweep (see Mitigation).

Cost to grief. Per seat: stranded fee = `min_balance(2 * Account::LEN) ≈ 0.0023 SOL`, plus realloc rent for `2 * GLOBAL_BLOCK_SIZE = 128` bytes ≈ `0.0009 SOL`. Total per seat ≈ `0.003 SOL`; 999 seats ≈ `3 SOL` (~$600 at ~$200/SOL). Scriptable, no signature races beyond winning the slot in which the LP global appears, and no economic payoff for the attacker (pure griefing or competitor-disruption).

Location

https://github.com/Bonasa-Tech/destiny/blob/9c541d0f7a06fa71f7c19b59a0b7bb649a38d901/programs/destiny/src/manifest.rs#L94-L134

https://github.com/Bonasa-Tech/destiny/blob/9c541d0f7a06fa71f7c19b59a0b7bb649a38d901/programs/destiny/src/processor/create_vault.rs#L100-L126

Relevant Code

```
pub fn sweep_token_to_global(
    &self,
    authority: &AccountInfo,
    system_program: &AccountInfo,
```

```
    signers: &[Signer],
) -> ProgramResult {
    // Ensure there's a global seat before trying to deposit.
    let should_create_seat = {
        let global = self.load_global_unchecked();
        if !global.has_global_seat(&(*authority.key()).into()) {
            if global.fixed.needs_eviction() {
                msg!("eviction not implemeted yet. skipping sweep to global");
                return Ok(());
            } else {
                true
            }
        } else {
            false
        }
    };
    if should_create_seat {
        // GlobalAddTrader CPI ...
    }

    let (balance, _) = get_token_balance_and_token_standard(&self.token, authority.key())?;
    if balance > 0 {
        self.deposit_global(balance, authority, signers)
    } else {
        Ok(())
    }
}
```

Mitigation Suggestion

Add a `GlobalEvict` wrapper to destiny. Mirror the existing `GlobalDeposit` / `GlobalWithdraw` wrappers in `programs/destiny/src/manifest.rs` with a `evict_global` helper that builds the `GlobalEvict` CPI with `authority` as the `payer` signer. Expose it on a trader-gated processor (e.g. inside `mint_lp` itself, conditionally on `needs_eviction()`, so each `mint_lp` call can opportunistically reclaim seats by depositing 1 atom more than the lowest-balance evictee). Because griefer seats hold zero atoms, 1 atom of LP suffices to displace one of them; one eviction per `mint_lp` is sufficient to recover, since after that authority holds a seat permanently and the LP path resumes normal operation.

Remediation

Fixed in https://github.com/Bonasa-Tech/destiny/pull/50 .

ACC-M3 Token program substitution allows timer reset without distribution

MEDIUM FIXED

Description

By pre-creating a Token-2022 ATA for (state_pda, rewards_mint) using the closeable-mint extension trick (create Token-2022 mint with closeable extension → create ATA for state program → close mint → recreate as regular mint), and then registering a regular token mint at the same mint address as the reward mint, the authority can log a 0-reward distribution that resets start_of_last_mint_timestamp without transferring any tokens, blocking legitimate distributions for 3600 seconds at will.

The root cause is that the ATA derivation at line 70 uses the caller-supplied `token_program_info.key()` instead of the token program associated with the state's `rewards_mint`. This allows substituting a Token-2022 ATA (with 0 balance) for the legitimate SPL Token ATA.

Location

https://github.com/Bonasa-Tech/destiny/blob/9c541d0f7a06fa71f7c19b59a0b7bb649a38d901/programs/fortune/src/processor/distribute_rewards.rs#L67-L78

Relevant Code

```
/// distribute_rewards.rs L67-L78
    let state_address: Address = Address::new_from_array(*state_info.key());
    let rewards_mint_address: Address = Address::new_from_array(state.rewards_mint);
    let token_program_address: Address = Address::new_from_array(*token_program_info.key());
    let expected_rewards_token_account: Address = get_associated_token_address_with_program_id(
        &state_address,
        &rewards_mint_address,
        &token_program_address,
    );
    if rewards_token_account_info.key().as_ref() != expected_rewards_token_account.as_ref() {
        return Err(ProgramError::InvalidSeeds);
    }
```

Mitigation Suggestion

Check that the provided token program matches the mint's token program on both initialization and distribution.

Remediation

Fixed in https://github.com/Bonasa-Tech/destiny/pull/46 .

ACC-M4 Free Eviction Enables Rent Griefing

MEDIUM FIXED

Description

We found that evicting users from global seats does not charge any fee, unlike acquiring a global seat, which requires paying rent for token accounts and increasing global seat size.

Because eviction is free:

- A malicious user can continuously evict normal users without cost.
- Close its token accounts.
- Victims may need to recreate token accounts for the attacker, enabling rent griefing attacks.

Location

https://github.com/Bonasa-Tech/manifest/blob/main/programs/manifest/src/program/processor/global_evict.rs

Mitigation Suggestion

Charge a fee for eviction similar to the cost of acquiring a global seat, or require the evictor to cover the rent/state cost.

Remediation

Fixed in https://github.com/Bonasa-Tech/manifest/pull/601 .

ACC-M5 Wrong PDA Seeds Break Global Account Recovery Path

MEDIUM FIXED

Description

We found that process_global_create uses the wrong PDA signing seeds when refunding lamports from an already-funded global account.

global_bump is derived for the global account PDA, but the code signs with global_vault_seeds_with_bump!, which derives the global vault PDA instead. Because these seeds produce a different PDA, invoke_signed fails with a PDA mismatch.

Location

https://github.com/Bonasa-Tech/manifest/blob/c56eafec41a075560404b52365291eb30e3c9ddf/programs/manifest/src/program/processor/global_create.rs#L61

Relevant Code

```
global_vault_seeds_with_bump!(global_mint.info.key, global_bump),
```

Mitigation Suggestion

Replace global_vault_seeds_with_bump!(global_mint.info.key, global_bump) with global_seeds_with_bump!(global_mint.info.key, global_bump).

Remediation

Fixed in https://github.com/Bonasa-Tech/manifest/pull/596 .

ACC-M6 AUM Desync Between Fortune and Destiny Can Over-Distribute Rewards

MEDIUM FIXED

Description

We found that Fortune assumes rewards are reflected in Destiny's AUM immediately, but this is not true. Fortune updates its own internal baseline after distribution, while Destiny's last_epoch_aum is only refreshed later during MintLP.

This creates a temporary inconsistency where:

- Fortune uses updated AUM (post-reward)
- Destiny still reports stale AUM (pre-reward)

If MintLP is not called between two distributions, Fortune will compare its updated baseline against a stale Destiny AUM and miscalculate rewards, leading to over-distribution.

Example:

- First distribution: AUM goes from 14 → 15 → target 17 → rewards distributed → Fortune baseline = 17
- MintLP not called → Destiny still reports AUM = 15
- Next distribution: Fortune compares 17 (baseline) vs 15 (stale) → incorrect calculation → excess rewards

Location

https://github.com/Bonasa-Tech/destiny/blob/66cfc067e5b27fdc3c2cb8ddef9550bdeeda9169/programs/fortune/src/processor/distribute_rewards.rs#L262

Relevant Code

```
state.start_of_last_mint_aum = u64::from(destiny_vault_state.last_epoch_aum) + reward_amount;
```

Mitigation Suggestion

Ensure AUM is synchronized before reward distribution (e.g., require a fresh AUM update or recompute AUM inside Fortune before using it).

Remediation

Fixed in https://github.com/Bonasa-Tech/destiny/pull/48 and https://github.com/Bonasa-Tech/destiny/pull/56 .

ACC-M7 Hardcoded 6-Decimal Assumption Can Break Reward Accounting

MEDIUM FIXED

Description

We found that the Fortune program assumes the quote token has 6 decimals. If a different token is used, this can cause incorrect reward accounting, including overpaying, underpaying, or even denial of service.

This assumption affects multiple parts of the logic, including:

- DEFAULT_MAX_REWARD_AMOUNT
- ABSOLUTE_MAX_CAP
- Token transfers (including Token-2022 flows)

Location

https://github.com/Bonasa-Tech/destiny/blob/66cfc067e5b27fdc3c2cb8ddef9550bdeeda9169/programs/fortune/src/state.rs#L50 https://github.com/Bonasa-Tech/destiny/blob/66cfc067e5b27fdc3c2cb8ddef9550bdeeda9169/programs/fortune/src/processor/distribute_rewards.rs#L195 https://github.com/Bonasa-Tech/destiny/blob/66cfc067e5b27fdc3c2cb8ddef9550bdeeda9169/programs/fortune/src/processor/distribute_rewards.rs#L246-L247

Relevant Code

```
pub const DEFAULT_MAX_REWARD_AMOUNT: u64 = 100_000_000;

    const ABSOLUTE_MAX_CAP: u64 = 200_000_000;

                // TODO: Fix this assumption
                decimals: 6,
```

Mitigation Suggestion

Adjust all reward and cap calculations based on the actual token decimals instead of assuming 6 decimals.

Remediation

Fixed in https://github.com/Bonasa-Tech/destiny/pull/34 .

ACC-M8 Zero start_of_last_mint_float Can DoS Reward Distribution

MEDIUM FIXED

Description

We found that the reward distribution instruction checks start_of_last_mint_float to prevent division by zero. However, when a new vault and distributor are created at the same time, this value is initialized to 0.

As a result, any attempt to run reward distribution will fail due to the division-by-zero protection, effectively causing a denial of service for the entire reward mechanism for that vault.

Location

https://github.com/Bonasa-Tech/destiny/blob/66cfc067e5b27fdc3c2cb8ddef9550bdeeda9169/programs/fortune/src/processor/initialize.rs#L54-L56

https://github.com/Bonasa-Tech/destiny/blob/66cfc067e5b27fdc3c2cb8ddef9550bdeeda9169/programs/fortune/src/processor/distribute_rewards.rs#L126-L128

Relevant Code

```
// Extract both AUM and float values from the vault
    let start_of_last_mint_aum: u64 = destiny_vault_state.last_epoch_aum.into();
    let start_of_last_mint_float: u64 = destiny_vault_state.last_epoch_float.into();


    // 4. Calculate AUM per float change
    let has_valid_float: bool = state.start_of_last_mint_float > 0;
    let start_aum_per_float: f64 = if has_valid_float {
```

Mitigation Suggestion

Disallow initialization of the distributor when start_of_last_mint_float == 0.

Remediation

Fixed in https://github.com/Bonasa-Tech/destiny/pull/33 .

ACC-M9 Fortune Rewards State Could Be Permanently Captured by a Frontrunner

MEDIUM FIXED

Description

We found that fortune initialize only checks whether authority_info signed, but it does not verify that this signer is the Destiny vault owner, trader, or any trusted admin. Since the rewards PDA is unique per vault, the first caller to initialize it becomes the stored authority forever.

As a result, any frontrunner could initialize the rewards state first, lock out the legitimate operator, and take control of all future admin-gated reward actions for that vault.

Location

https://github.com/Bonasa-Tech/destiny/blob/66cfc067e5b27fdc3c2cb8ddef9550bdeeda9169/programs/fortune/src/processor/initialize.rs#L58-L61

Relevant Code

```
let (expected_state_key, state_bump): (Pubkey, u8) = pinocchio::pubkey::find_program_address(
        &[SEED_PREFIX_REWARDS_STATE, destiny_vault_info.key().as_ref()],
        program_id,
    );
```

Mitigation Suggestion

Add the authority to the PDA seeds so the intended authority can initialize and control the rewards state.

Remediation

Fixed in https://github.com/Bonasa-Tech/destiny/pull/28 and https://github.com/Bonasa-Tech/destiny/pull/42 .

ACC-M10 Off-Chain Signers Can Be Forwarded as CPI Signers

MEDIUM FIXED

Description

We found that in the new design, the trader is a PDA of the Fate program, and transactions are signed off-chain and then executed via CPI from Fate to Destiny.

During this CPI, the program forwards PDA seeds and also passes through any signer accounts present in the CPI account list. The issue is that if signer A or signer B is included in the CPI accounts, they are forwarded as signers as well.

This means off-chain signer accounts can unintentionally become signers in the CPI context. Since the target program is not hardcoded, this exposes those signer privileges to arbitrary programs, which may misuse them for unintended or unauthorized actions.

Location

https://github.com/Bonasa-Tech/destiny/blob/66cfc067e5b27fdc3c2cb8ddef9550bdeeda9169/programs/fate/src/processor.rs#L67-L68

Relevant Code

```
.map(|(i, acc)| {
            let is_signer = i == pda_idx || acc.is_signer();
```

Mitigation Suggestion

We should explicitly check CPI accounts and, if an account is signer A or signer B, pass it as a non-signer in the CPI.

Remediation

Fixed in https://github.com/Bonasa-Tech/destiny/pull/35 .

ACC-L1 `manifest_batch_update` forwards `instruction_data` verbatim with no discriminator pin

LOW FIXED

Description

`process_manifest_batch_update` (manifest_batch_update.rs:82-90) takes the trader's `instruction_data` and passes it directly as the `data` field of a manifest CPI built by `MarketAccountInfos::batch_update` (manifest.rs:265-294). The wrapper hardcodes the account list (13 slots in BatchUpdate's exact layout, with `authority` PDA-signing) but does not enforce that `instruction_data[0]` equals `ManifestInstruction::BatchUpdate as u8` (= 6). Whichever discriminator the trader puts in the first byte is what manifest dispatches.

This is currently safe in practice because manifest's per-instruction account loaders reject the BatchUpdate-shaped account list for almost every other instruction:

- `ClaimSeat (1)`: account positions match (payer/market/system), but `claim_seat` errors `AlreadyClaimedSeat` since `list_market` already claimed authority's seat (list_market.rs:84-105). Benign.
- `Expand (5)`: account positions match. Succeeds: grows the market by one block, paying authority's lamports. The lamports come from authority's prefund, which the trader's signer paid in this same tx, so the trader is wasting their own SOL. Benign.
- `Deposit (2)` / `Withdraw (3)`: position 2 expects a token account; destiny supplies `system_program`. `TokenAccountInfo` loader rejects.
- `Swap (4)` / `SwapV2 (13)`: position 3 expects `trader_base` token account; destiny supplies `base.mint`. Loader rejects (or signer check on SwapV2's position 1).
- `CreateMarket (0)`: vault PDAs at positions 5/6 must be uninitialised; destiny supplies an initialised market vault. Rejects.
- `GlobalCreate / GlobalAddTrader / GlobalDeposit / GlobalWithdraw / GlobalEvict (7..11)`: position 1 expects a manifest-owned `global` (GlobalFixed discriminant); destiny supplies the `market` (different discriminant). Rejects.
- `GlobalClean (12)`: position 3 expects a `global`; destiny supplies `base.mint`. Rejects.

So the only "extra" capability the trader can actually unlock through the passthrough is `Expand`, and the trader pays for it themselves. No current security boundary breach.

The risk is forward-looking: any future manifest version that adds a new instruction at a new discriminator value with an account layout that overlaps BatchUpdate's would automatically become callable through destiny's passthrough with `authority`-PDA signing rights. Since destiny upgrades to manifest's SDK independently of manifest's own development cadence (see commit `479cee7` "Update to newer manifest sdk"), an adversarial-by-accident composition window exists.

The same passthrough pattern does not appear elsewhere in destiny: `manifest_cancel_all`, `manifest_rebalance`, `mint_lp`, `list_market`, and all global wrappers in `manifest.rs` build their own data buffers with hardcoded discriminators. `manifest_batch_update` is the unique trader-controlled passthrough.

Location

https://github.com/Bonasa-Tech/destiny/blob/9c541d0f7a06fa71f7c19b59a0b7bb649a38d901/programs/destiny/src/processor/manifest_batch_update.rs#L82-L90

Relevant Code

```
/// programs/destiny/src/processor/manifest_batch_update.rs L82-L90
msg!("batch update");
market_ais.batch_update(
```

```
    base_ais,
    quote_ais,
    instruction_data,  // <-- forwarded verbatim, no discriminator check
    authority,
    system_program,
    &authority_signers,
)?;
<...>
/// programs/destiny/src/manifest.rs L265-L294
pub fn batch_update(
    &self,
    base: &BalanceAccountInfos,
    quote: &BalanceAccountInfos,
    data: &[u8],  // <-- placed straight into Instruction.data
    authority: &AccountInfo,
    system_program: &AccountInfo,
    signers: &[Signer],
) -> ProgramResult {
    let ix = Instruction {
        program_id: manifest::ID.as_array(),
        data,
        accounts: &[ /* fixed BatchUpdate layout, 13 accounts */ ],
    };
    invoke_signed(&ix, &[ /* same 13 accounts */ ], signers)
}
```

Mitigation Suggestion

Add a single byte check at the entry of `process_manifest_batch_update` (or inside the `batch_update` wrapper) before forwarding to manifest:

```
let Some((&disc, batch_data)) = instruction_data.split_first() else {
    return Err(ProgramError::InvalidInstructionData);
};
if disc != ManifestInstruction::BatchUpdate as u8 {
    return Err(ProgramError::InvalidInstructionData);
}
```

Then prepend the discriminator inside the wrapper, so callers pass only the BatchUpdate body. This pins destiny's intent at the instruction boundary and makes the passthrough robust against future manifest additions.

Remediation

Fixed in https://github.com/Bonasa-Tech/destiny/pull/52 .

ACC-L2 Destiny `mint_lp` transfers profit-share LP to an unvalidated `profit_share_token`

LOW FIXED

Description

The recipient of the 20% profit-share LP transfer is passed in as an account and never checked: no stored config, no ATA derivation, no owner check. Whoever calls `mint_lp` (the signer, verified as `trader`) picks the destination.

Location

https://github.com/Bonasa-Tech/destiny/blob/9c541d0f7a06fa71f7c19b59a0b7bb649a38d901/programs/destiny/src/processor/mint_lp.rs#L85-L95

https://github.com/Bonasa-Tech/destiny/blob/9c541d0f7a06fa71f7c19b59a0b7bb649a38d901/programs/destiny/src/processor/mint_lp.rs#L225-L239

Relevant Code

```
/// programs/destiny/src/processor/mint_lp.rs L85-L95
pub fn process_mint_lp(
    program_id: &Pubkey,
    accounts: &[AccountInfo],
    instruction_data: &[u8],
) -> ProgramResult {
    // Destructure accounts slice
    let [signer, vault, authority, quote_mint, profit_share_token, manifest_program, system_program, remainin
g_accounts @ ..] =
        accounts
    else {
        return Err(ProgramError::NotEnoughAccountKeys);
    };
<...>
/// programs/destiny/src/processor/mint_lp.rs L225-L239
if update.profit_issuance > BaseAtoms::ZERO {
    msg!("pay out profit share");
    msg!("withdraw lp global if possible");
    lp_account_infos.withdraw_global_if_possible(authority, &authority_signers)?;
    msg!("transfer profit share");
    Transfer {
        from: lp_account_infos.token(),
        to: profit_share_token,
        authority,
        amount: update.profit_issuance.as_u64(),
    }
    .invoke_signed(&authority_signers)?;
jj
```

Mitigation Suggestion

Persist the expected profit-share recipient in `VaultState` at vault creation (settable only by `owner`, not `trader`). Enforce `profit_share_token.key() == vault_state.profit_share_token` (or that it is the ATA of a stored owner).

Remediation

Fixed in https://github.com/Bonasa-Tech/destiny/pull/54 .

ACC-L3 Destiny `set_owner` / `set_trader` have no two-step handover

LOW ACKNOWLEDGED

Description

`process_set_owner` writes the raw `new_owner.key()` into `vault_state.owner` with no signer requirement on the new principal and no pending/accept flow. A mistyped key, or a key whose secret is not in fact held by the intended recipient, permanently bricks owner-only operations (`list_asset`, `list_market`, `set_owner`, `set_trader`, `create_token_metadata`). Because the vault is a program-owned account, recovery would require a destiny program upgrade that adds an override path. The same shape applies to `set_trader`: a typo there is recoverable by the still-valid owner, so the risk is lower.

Location

https://github.com/Bonasa-Tech/destiny/blob/9c541d0f7a06fa71f7c19b59a0b7bb649a38d901/programs/destiny/src/processor/set_owner.rs#L7-L24

Relevant Code

```
/// programs/destiny/src/processor/set_owner.rs L7-L24
pub fn process_set_owner(
    program_id: &Pubkey,
    accounts: &[AccountInfo],
    _instruction_data: &[u8],
) -> ProgramResult {
    let [owner_signer, vault, authority, quote_mint, new_owner, ..] = accounts else {
        return Err(ProgramError::NotEnoughAccountKeys);
    };

    VaultState::verify_account_infos(vault, authority, quote_mint, program_id)?;
    let vault_state = VaultState::load_mut_checked(vault)?;
    vault_state.verify_owner(owner_signer)?;

    vault_state.owner = *new_owner.key();

    Ok(())
}
```

Mitigation Suggestion

Split ownership transfer into two instructions:

• `propose_owner`: current owner signs, writes `pending_owner`.

• `accept_owner`: `pending_owner` signs, atomically promotes to `owner`.

This prevents typo-bricking and ensures the new principal proves key possession before the transfer commits.

Remediation

Issue has been acknowledged.

Client:

Vault creation & configuration will happen on development machine: owner and trader have stricter security requirements. (fate / squads). program upgrade recovery is nearly as complex as requesting a signature from owner, so the cost doesn't outweigh the benefit here.

ACC-L4 mint_lp Does Not Reduce Vault LP Balance After Sending Trader Share

LOW FIXED

Description

We found that in mint_lp, after calculating the trader's LP share and transferring it to the trader, the vault's balance_cash for LP is not reduced.

These LP tokens no longer belong to the vault, but the internal accounting still counts them as if they do. As a result, balance_cash and lp_token_balance can become higher than the real vault LP balance, leading to outdated and incorrect accounting.

Location

https://github.com/Bonasa-Tech/destiny/blob/2460e0600f0ff040f7960ed220e21e66a7af071e/programs/destiny/src/processor/mint_lp.rs#L223-L237

Relevant Code

```
if update.profit_issuance > BaseAtoms::ZERO {
        msg!("pay out profit share");
        msg!("withdraw lp global if possible");
        lp_account_infos.withdraw_global_if_possible(authority, &authority_signers)?;
        msg!("transfer profit share");
        Transfer {
            from: lp_account_infos.token(),
            to: profit_share_token,
            authority,
            amount: update.profit_issuance.as_u64(),
        }
        .invoke_signed(&authority_signers)?;
        msg!("sweep lp to global");
        lp_account_infos.sweep_token_to_global(authority, system_program, &authority_signers)?;
    }
```

Mitigation Suggestion

Reduce the vault's LP balance_cash by the profit_issuance immediately after transferring the LP tokens.

Remediation

Fixed in https://github.com/Bonasa-Tech/destiny/pull/44 .

ACC-L5 Token-2022 Transfer Fees Cause Reward Accounting Mismatch

LOW FIXED

Description

We found that the Fortune program supports Token-2022, but the reward distribution logic does not account for transfer fees. When distributing rewards, the program updates its internal state assuming the full reward amount is transferred, but due to transfer fees, the actual tokens received in the quote vault can be less than expected.

This creates an inconsistency between:

- Fortune's internal accounting (records full reward), and
- Actual vault balance (net amount after fees).

Location

https://github.com/Bonasa-Tech/destiny/blob/2460e0600f0ff040f7960ed220e21e66a7af071e/programs/fortune/src/processor/distribute_rewards.rs#L278

Relevant Code

```
state.start_of_last_mint_aum = u64::from(destiny_vault_state.last_epoch_aum) + reward_amount;
```

Mitigation Suggestion

Base state updates on the actual received token amount (post-fee) instead of the intended transfer amount.

Remediation

Fixed in https://github.com/Bonasa-Tech/destiny/pull/38 .

ACC-L6 Delayed Reward Processing Can Underpay Rewards

LOW ACKNOWLEDGED

Description

We found that the reward distribution max amount assumes rewards are distributed every hour. If distribution is delayed (for example, called after 2 hours), the max reward does not scale with the elapsed time, so the user receives less reward than intended.

Location

https://github.com/Bonasa-Tech/destiny/blob/66cfc067e5b27fdc3c2cb8ddef9550bdeeda9169/programs/fortune/src/processor/distribute_rewards.rs#L190-L203

Relevant Code

```
// Transfer the minimum of desired reward amount and available balance
    // This prevents insufficient funds errors while respecting APY calculations
    // Cap it at the configured max_reward_amount per hour (default $100 for 6 decimal tokens).
    // Use default if max_reward_amount is 0.
    // Sanity check: never exceed 200 USDC regardless of config.
    const ABSOLUTE_MAX_CAP: u64 = 200_000_000;
    let max_reward_cap: u64 = if state.max_reward_amount == 0 {
        RewardsState::DEFAULT_MAX_REWARD_AMOUNT
    } else {
        state.max_reward_amount.min(ABSOLUTE_MAX_CAP)
    };
    let reward_amount: u64 = desired_reward_amount
        .min(available_balance)
        .min(max_reward_cap);
```

Mitigation Suggestion

Scale the max reward amount by the number of hours elapsed since the last distribution.

Remediation

The issue acknowledged, and the comments updated https://github.com/Bonasa-Tech/destiny/pull/36 , the max reward is not per hour its per distribution.

ACC-L7 Reward Distributor Missing Vault Token Account Validation

LOW FIXED

Description

We found that the reward distributor does not validate vault_token_account_info. As a result, if the authority key is compromised, an attacker can redirect reward distributions to any arbitrary token account instead of the intended vault.

This breaks the assumption that rewards are always sent to the correct vault-controlled account and enables unauthorized redirection of funds.

Location

https://github.com/Bonasa-Tech/destiny/blob/66cfc067e5b27fdc3c2cb8ddef9550bdeeda9169/programs/fortune/src/processor/distribute_rewards.rs#L232-L250

Relevant Code

```
if pubkey_eq(token_program_info.key(), &pinocchio_token::ID) {
            let transfer_instruction = TokenTransfer {
                from: rewards_token_account_info,
                to: vault_token_account_info,
                amount: reward_amount,
                authority: state_info,
            };
            transfer_instruction.invoke_signed(&state_signers)?;
        } else if pubkey_eq(token_program_info.key(), &pinocchio_token_2022::ID) {
            let transfer_instruction = Token2022Transfer {
                from: rewards_token_account_info,
                to: vault_token_account_info,
                mint: mint_account_info,
                amount: reward_amount,
                // TODO: Fix this assumption
                decimals: 6,
                authority: state_info,
                token_program: token_program_info.key(),
            };
```

Mitigation Suggestion

Validate that vault_token_account_info matches the expected vault token account to prevent arbitrary redirection. However, this introduces a trade-off: the authority will no longer be able to redirect or withdraw undistributed rewards to alternative accounts (e.g., for recovery or migration).

A balanced approach is to enforce strict validation for normal reward distribution, while providing a separate, explicitly authorized instruction (e.g., admin-only “recover/withdraw undistributed rewards”) that allows controlled flexibility without weakening the main flow.

Remediation

Fixed in https://github.com/Bonasa-Tech/destiny/pull/29 .

ACC-I1 Unguarded debug `msg!` / `log!` spam across hot paths

INFO FIXED

Description

118 `msg!` / `log!` call sites across 14 files, none gated by `#[cfg(debug_assertions)]`, a `feature = "debug"` flag, or an off-chain verbosity level. The density concentrates in the fortune hot paths: `distribute_rewards.rs` (27 sites) and `initialize.rs` (19 sites), where every call fires on every successful transaction. Representative sample:

```
// programs/fortune/src/processor/distribute_rewards.rs L96-L118
log!("Current AUM {}", current_aum);
log!("Start of last mint AUM {}", state.start_of_last_mint_aum);
log!("Current Float {}", current_float);
log!("Start of last mint Float {}", state.start_of_last_mint_float);
// ...
log!("Current timestamp {}", current_timestamp);
log!("Start of last mint timestamp {}", state.start_of_last_mint_timestamp);
// ...
log!("Time elapsed seconds {}", time_elapsed_seconds as u64);
log!("Time elapsed years {}", time_elapsed_years as u64);
```

Each `msg!` / `log!` costs compute units (`msg!` is ~100 CU per call plus bytes cost; `pinocchio_log::log!` is cheaper but still non-zero). Several hundred CUs per instruction is not catastrophic (fortune's distribute_rewards is currently at ~18.5k / 200k CUs per the test output) but is pure overhead once the program is stable, shows up as noise in indexers and RPC logs, and leaks internal state field names / values to anyone reading transaction logs.

Location

https://github.com/Bonasa-Tech/destiny/blob/9c541d0f7a06fa71f7c19b59a0b7bb649a38d901/programs/fortune/src/processor/distribute_rewards.rs#L96-L118

Also affected (per-file counts of `msg!` + `log!`):

| File | Count | |------|-------| | `programs/fortune/src/processor/distribute_rewards.rs` | 27 | | `programs/fortune/src/processor/initialize.rs` | 19 | | `programs/destiny/src/processor/mint_lp.rs` | 15 | | `programs/destiny/src/processor/manifest_rebalance.rs` | 9 | | `programs/destiny/src/token.rs` | 8 | | `programs/destiny/src/processor/create_vault.rs` | 7 | | `programs/destiny/src/processor/manifest_cancel_all.rs` | 7 | | `programs/destiny/src/processor/migrate_vault.rs` | 6 | | `programs/destiny/src/manifest.rs` | 5 | | `programs/destiny/src/state.rs` | 4 | | `programs/fate/src/processor.rs` | 3 | | `programs/destiny/src/processor/create_token_metadata.rs` | 3 | | `programs/destiny/src/processor/list_market.rs` | 3 | | `programs/destiny/src/processor/manifest_batch_update.rs` | 2 |

Relevant Code

Mitigation Suggestion

Gate the trace-level logs behind a cargo feature so they compile out in release:

```
# Cargo.toml
[features]
default = []
trace-logs = []
```

```
// replace
log!("Current AUM {}", current_aum);
// with
#[cfg(feature = "trace-logs")]
log!("Current AUM {}", current_aum);
```

Keep error / state-transition `msg!`s (e.g. the ones that precede a returned `Err` or log "claim seat" / "cancel all" at the boundaries of a CPI), those are useful in live debugging. The ones that dump every field value of in-memory state can go.

Remediation

Fixed in https://github.com/Bonasa-Tech/destiny/pull/53 .

ACC-I2 `VaultState.state_bump` is actually the vault PDA bump (naming drift)

INFO FIXED

Description

The `VaultState::state_bump` field is used everywhere as the bump for the vault PDA. The seed pair is always `SEED_PREFIX_VAULT = b"vaul"`:

```
// state.rs:99-104 (verify_account_infos)
let expected_vault_pda = derive_address(
    &[SEED_PREFIX_VAULT, &DEPLOYER, quote_mint.key()],
    Some(vault_state.state_bump),
    program_id,
);
```

And `create_vault.rs:64-67` derives it with the local name `vault_bump`, then writes it into the `state_bump` field. The point of view at write time is clearly "this is the vault bump." There is no separate "state" PDA in destiny; `DestinyAccount` only enumerates `Vault` (and `Uninitialized`), so the field's label refers to a concept that doesn't exist in the program.

Location

https://github.com/Bonasa-Tech/destiny/blob/9c541d0f7a06fa71f7c19b59a0b7bb649a38d901/programs/destiny/src/state.rs#L27-L40

Relevant Code

```
/// programs/destiny/src/state.rs L27-L40
#[repr(C)]
#[derive(Clone)]
pub struct VaultState {
    /// Discriminant for identifying this type of account.
    pub discriminant: DestinyAccount,
    pub version: u8,
    pub state_bump: u8,
    pub authority_bump: u8,
    pub high_watermark_lp_token_price_matissa: u32,
    pub last_epoch_aum: QuoteAtoms,
    pub last_epoch_float: BaseAtoms,
    pub balance_caches: Vec<BalanceCache, MAX_BALANCES>,
    pub market_caches: Vec<MarketCache, MAX_MARKETS>,
    pub owner: Pubkey,
    pub trader: Pubkey,
}
```

Mitigation Suggestion

Rename to `vault_bump` the next time a layout-breaking change ships (e.g. a future migration). A standalone rename forces a client / IDL / fixture cascade that costs more than it returns.

Remediation

Fixed in https://github.com/Bonasa-Tech/destiny/pull/49 .

ACC-I3 Fate could be called as CPI and signature assembled

INFO FIXED

Description

If both signers ever sign a transaction together calling program X, and this program X for example was upgraded then maliciously calls Fate, it can use their signatures to create a fate PDA signature and an arbitrary CPI.

Signer A+B --> Program X --> Fate --> Arbitrary CPI

This is possible because Fate allows to be called as CPI.

Mitigation Suggestion

Add a check to fate that it is not called as a CPI call. (Note that this would break structures where one of the signers is an on-chain multisig account)

Remediation

Fixed in https://github.com/Bonasa-Tech/destiny/pull/47 .

ACC-I4 Wrong Offset Used for Remaining Accounts in split_manifest_market_account_infos

INFO FIXED

Description

We found that split_manifest_market_account_infos() in manifest.rs calculates the returned remaining accounts slice using the wrong offset. The first three returned account groups are parsed correctly, but the trailing slice starts only after the market-account group, instead of starting after both balance groups and the market group.

This does not appear to be exploitable today because current call sites ignore the fourth returned value. However, it is still a latent issue and a maintenance footgun.

Location

https://github.com/Bonasa-Tech/destiny/blob/2460e0600f0ff040f7960ed220e21e66a7af071e/programs/destiny/src/manifest.rs#L584-L605

Relevant Code

```
pub fn split_manifest_market_account_infos(
    account_infos: &[AccountInfo],
) -> Result<
    (
        &BalanceAccountInfos,
        &BalanceAccountInfos,
        &MarketAccountInfos,
        &[AccountInfo],
    ),
    ProgramError,
> {
    let (balance_accounts, remaining_accounts) = split_balance_account_infos(2, account_infos)?;
    if remaining_accounts.len() < MarketAccountInfos::NUM_ACCOUNTS {
        return Err(ProgramError::NotEnoughAccountKeys);
    }
    let market_accounts = unsafe { &*(remaining_accounts.as_ptr().cast::<MarketAccountInfos>()) };
    let remaining_accounts = unsafe {
        &*(slice_from_raw_parts(
            account_infos.as_ptr().add(MarketAccountInfos::NUM_ACCOUNTS),
            account_infos.len() - MarketAccountInfos::NUM_ACCOUNTS,
        ))
    };
```

Mitigation Suggestion

Compute the remaining-accounts slice from the full consumed offset, after both balance groups and the market group.

Remediation

Fixed in https://github.com/Bonasa-Tech/destiny/pull/41 .

ACC-I5 APY vs APR Mislabeling in target_apy_bps

INFO FIXED

Description

The target_apy_bps field is actually being used as an APR, not an APY.

The formula 1 + (rate * time) represents simple interest (APR), not compound interest (APY). However, the field name and surrounding documentation refer to it as APY, which is misleading.

Location

https://github.com/Bonasa-Tech/destiny/blob/66cfc067e5b27fdc3c2cb8ddef9550bdeeda9169/programs/fortune/src/state.rs#L35-L36

Relevant Code

```
/// Target annual percentage rate in basis points (10000 = 100%, 1000 = 10%)
    pub target_apy_bps: u16,
```

Mitigation Suggestion

Check what is intended and rewrite the name if APR is the intended one.

Remediation

Fixed in https://github.com/Bonasa-Tech/destiny/pull/32 .

ACC-I6 Wrong Rewards Mint Could Be Initialized and Break Reward Distribution

INFO FIXED

Description

We found that fortune initialize does not validate rewards_mint_info against the Destiny vault's expected quote mint. Since there can only be one reward distributor per destiny_vault, initializing it with the wrong mint can permanently block the intended setup and cause a denial of service.

Location

https://github.com/Bonasa-Tech/destiny/blob/66cfc067e5b27fdc3c2cb8ddef9550bdeeda9169/programs/fortune/src/processor/initialize.rs#L105

Relevant Code

Mitigation Suggestion

Validate rewards_mint_info against the vault's quote token mint before initialization.

Remediation

Fixed in https://github.com/Bonasa-Tech/destiny/pull/30 .

ACC-I7 Fortune Initialize Relies on Side Effect Instead of Rejecting Reinitialization

INFO FIXED

Description

We found that fortune initialize does not explicitly check whether the state is already initialized. After the first initialization, the account is owned by the Fortune program instead of the system program, so a later CPI to the system program for lamport transfer will fail.

This means reinitialization is blocked only because of system program behavior, not because the program clearly rejects it. This is fragile and makes the initialization logic less clear and harder to maintain.

Location

https://github.com/Bonasa-Tech/destiny/blob/66cfc067e5b27fdc3c2cb8ddef9550bdeeda9169/programs/fortune/src/processor/initialize.rs#L132-L140

Relevant Code

```
if current_lamports > 0 {
        msg!("Clearing squatted account");
        Transfer {
            from: state_info,
            to: authority_info,
            lamports: current_lamports,
        }
        .invoke_signed(&state_signers)?;
    }
```

Mitigation Suggestion

Explicitly check whether the state is already initialized and reject reinitialization early.

Remediation

Fixed in https://github.com/Bonasa-Tech/destiny/pull/31 .

APPENDIX

Vulnerability Classification

We rate our issues according to the following scale. Informational issues are reported informally to the developers and are not included in this report.

Severity	Description
Critical	Vulnerabilities that can be easily exploited and result in loss of user funds, or directly violate the protocol's integrity. Immediate action is required.
High	Vulnerabilities that can lead to loss of user funds under non-trivial preconditions, loss of fees, or permanent denial of service that requires a program upgrade. These issues require attention and should be resolved in the short term.
Medium	Vulnerabilities that may be more difficult to exploit but could still lead to some compromise of the system's functionality. For example, partial denial of service attacks, or such attacks that do not require a program upgrade to resolve, but may require manual intervention. These issues should be addressed as part of the normal development cycle.
Low	Vulnerabilities that have a minimal impact on the system's operations and can be fixed over time. These issues may include inconsistencies in state, or require such high capital investments that they are not exploitable profitably.
Informational	Findings that do not pose an immediate risk but could affect the system's efficiency, maintainability, or best practices.

Audit Methodology

Accretion is a boutique security auditor specializing in Solana's ecosystem. We employ a customized approach for each client, strategically allocating our resources to maximize code review effectiveness. Our auditors dedicate substantial time to developing a comprehensive understanding of each program under review, examining design decisions, expected and edge-case behaviors, invariants, optimizations, and data structures, while meticulously verifying mathematical correctness—all within the context of the developers' intentions.

Our audit scope extends beyond on-chain components to include associated infrastructure, such as user interfaces and supporting systems. Every audit encompasses both a holistic protocol design review and detailed line-by-line code analysis.

During our assessment, we focus on identifying:
• Solana-specific vulnerabilities
• Access control issues
• Arithmetic errors and precision loss
• Race conditions and MEV opportunities
• Logic errors and edge cases
• Performance optimization opportunities
• Invariant violations
• Account confusion vulnerabilities
• Authority check omissions
• Token22 implementation risks and SPL-related pitfalls
• Deviations from best practices

Our approach transcends conventional vulnerability classifications. We continuously conduct ecosystem-wide security research to identify and mitigate emerging threat vectors, ensuring our audits remain at the forefront of Solana security practices.