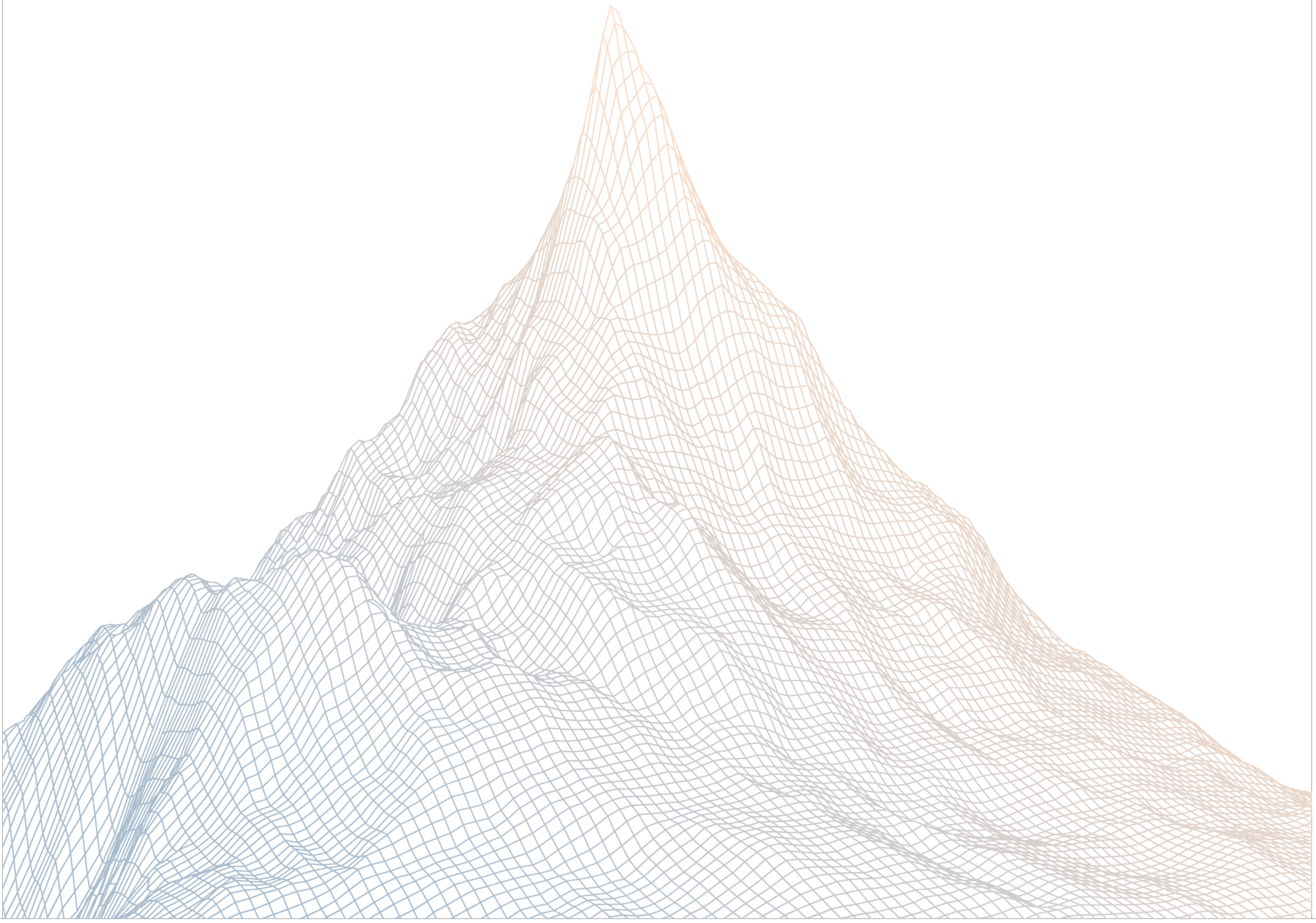
Zenith

America.Fun

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES:	December 17th to December 24th, 2025
AUDITED BY:	Mario Poneder peakbolt

Contents

1 Introduction 2

1.1 About Zenith 3

1.2 Disclaimer 3

1.3 Risk Classification 3

2 Executive Summary 3

2.1 About America.Fun 4

2.2 Scope 4

2.3 Audit Timeline 5

2.4 Issues Found 5

3 Findings Summary 5

4 Findings 8

4.1 Critical Risk 9

4.2 High Risk 11

4.3 Medium Risk 15

4.4 Low Risk 29

4.5 Informational 48

1 Introduction

1.1 About Zenith

Zenith assembles auditors with proven track records: finding critical vulnerabilities in public audit competitions.

Our audits are carried out by a curated team of the industry's top-performing security researchers, selected for your specific codebase, security needs, and budget.

Learn more about us at https://zenith.security.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an "as-is" and "as-available" basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

SEVERITY LEVEL	IMPACT: HIGH	IMPACT: MEDIUM	IMPACT: LOW
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2 Executive Summary

2.1 About America.Fun

America.Fun is a trusted crypto launchpad on Solana that makes it simple for anyone to create and launch their own token. Every token launched on the platform is paired with USD1, the stablecoin issued by World Liberty Financial.

The platform's native token, $AOL, is built directly into the ecosystem and rewards long-term stakers with a share of platform revenue.

2.2 Scope

The engagement involved a review of the following targets:

Target	staking-americafun-backend
Repository	https://github.com/americafun/staking-americafun-backend
Commit Hash	`323dfb284a92894cdb25f611c1f61d89e4e31c62`
Files	`programs/aol-staking-program/src/lib.rs`

2.3 Audit Timeline

December 17, 2025	Audit start
December 24, 2025	Audit end
January 7, 2026	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	1
High Risk	2
Medium Risk	10
Low Risk	15
Informational	10
Total Issues	**38**

3 Findings Summary

ID	Description	Status
C-1	claim_rewards allows multi-counting the same vault via duplicate remaining accounts	Resolved
H-1	Early unlock minimum not enforced on chain for early_unlock_vault instruction	Resolved
H-2	Old unfinalized snapshot can prematurely sweep rewards for newer snapshots	Resolved
M-1	No partial user snapshot state: Users can permanently underclaim if they have to omit vaults	Resolved
M-2	withdraw_vault, early_unlock_vault, and admin_force_unstake can be DoS by donation attack	Resolved
M-3	Remove admin_close_all_configs and admin_empty_protocol_accounts in production code	Resolved
M-4	Incorrect close pattern in admin_force_unstake causes runtime failure	Resolved
M-5	create_vault will be DoS for user after multiple creations due to overflow in total_vaults_created	Resolved
M-6	Events emit stale token balances after CPIs	Resolved
M-7	Timing invariant (snapshot vs. claim window) is not enforced	Resolved
M-8	execute_daily_snapshot fails to prevent negative/old/future snapshot_timestamp	Resolved
M-9	get_user_portfolio and get_user_claimable cannot handle ABSOLUTE_MAX_VAULTS_PER_USER due to Solana message size limit	Resolved
M-10	Hardcoded Rapid Testing should be removed for production	Resolved
L-1	Generic token account constraints allows caller to send penalty and swept rewards to arbitrary accounts	Resolved
L-2	Lack of validation in update_lock_tiers and update_penalty_tiers	Resolved
L-3	Misleading “Grace Period” error and timing semantics around claim and sweep	Resolved

ID	Description	Status
L-4	Loose validation of user_snapshot account in get_user_claimable instruction	Resolved
L-5	Unused limit constants and errors give a false sense of protection	Resolved
L-6	StakingVault.total_claimed and last_claim_at are never updated leading to incorrect get_vault_info.total_claimed	Resolved
L-7	early_unlock_min_seconds inconsistency and one□way behavior	Resolved
L-8	get_user_portfolio and get_user_claimable can multi-count vaults via duplicate remaining accounts	Resolved
L-9	Early-withdrawal with penalty still allowed after vault unlock time	Resolved
L-10	emit!() and msg!() can be truncated by RPC due to log limits	Acknowledged
L-11	Unrealistic MAX_VAULTS_PER_SNAPSHOT / MAX_VAULTS_PER_CLAIM constants vs Solana transaction limits	Resolved
L-12	Using msg()! in for loop within claim_rewards() could use excessive CU and fail	Resolved
L-13	Incorrect intermediary_transferred emitted in SnapshotExecutedEvent	Resolved
L-14	sweep_unclaimed_rewards should check for current_time > sweep_time instead	Resolved
L-15	claim_rewards should validate !snapshot.is_finalized	Resolved
I-1	Comprehensive test suite is out-of-sync with program behavior and not reliable as shipped	Resolved
I-2	Hardcoded test success summary printed even when tests fail	Resolved
I-3	Unused GlobalConfig.snapshot_counter field	Resolved
I-4	Redundant staking_bucket account in get_user_claimable context	Resolved

ID	Description	Status
I-5	Centralization risk: Admin can unilaterally drain all reward buckets	Acknowledged
I-6	Legacy sweep grace-period setting is unused code	Resolved
I-7	Initialization race: Anyone can become admin if protocol is not initialized immediately	Resolved
I-8	Redundant vault_registry binding in reward/portfolio logic	Resolved
I-9	Unused UserClosedAllVaultsEvent event definition	Resolved
I-10	BASIS_POINTS_DIVISOR name is strongly misleading	Resolved

4 Findings

4.1 Critical Risk

A total of 1 critical risk findings were identified.

[C-1] `claim_rewards` allows multi-counting the same vault via duplicate remaining accounts

SEVERITY: Critical	IMPACT: High
STATUS: Resolved	LIKELIHOOD: High

Target

- programs/aol-staking-program/src/lib.rs#L488-L505

Description:

The `claim_rewards` instruction computes a user's weight for a snapshot by iterating over `ctx.remaining_accounts` and summing the weights of any `StakingVault` accounts it finds:

```
msg!("[claim_rewards] Calculating user weight at snapshot time from {} vault
    accounts", ctx.remaining_accounts.len());

for acc_info in ctx.remaining_accounts.iter() {
    if let Ok(vault_data) = validate_vault_account(acc_info,
    Some(&user_key)) {
        // CRITICAL: Only count vaults created on or before snapshot
    timestamp
        if vault_data.locked_at <= snapshot.snapshot_timestamp {
            if let Ok(weight) = calculate_vault_weight(
                vault_data.amount_staked,
                vault_data.weight_multiplier_bp,
            ) {
                total_user_weight
    = total_user_weight.saturating_add(weight);
                msg!("[claim_rewards] Vault {} qualifies: locked_at={},
    weight={}",
                     vault_data.vault_index, vault_data.locked_at, weight);
            }
        }
    }
}
```

On Solana, an instruction's `accounts` list can reference the same account index multiple times. Anchor builds `ctx.remaining_accounts` directly from that list, so a malicious client can include the same vault PDA repeatedly in `remaining_accounts`. Since `validate_vault_account` only checks that each `acc_info` is a valid vault PDA for the user (and does not deduplicate vaults), the same vault's weight will be added once per occurrence. This allows an attacker to inflate `total_user_weight` nearly arbitrarily (up to `snapshot.total_global_weight`), and thus claim a disproportionately large share of the snapshot's `initial_staking_balance`, potentially draining most or all rewards for that snapshot.

Recommendations:

It is recommended to ensure each vault is only counted once per claim, for example track seen vaults in-memory (e.g., by `vault.key()`) and skip duplicates:

```
let mut seen_vaults = Vec::new();
for acc_info in ctx.remaining_accounts.iter() {
    if seen_vaults.contains(&acc_info.key()) {
        continue;
    }
    seen_vaults.push(acc_info.key());
    // validate_vault_account and add weight...
}
```

America.fun: Resolved with `@18fba21997 ...`

Zenith: Verified.

4.2 High Risk

A total of 2 high risk findings were identified.

[H-1] Early unlock minimum not enforced on□chain for `early_unlock_vault` instruction

`SEVERITY:` High	`IMPACT:` Medium
`STATUS:` Resolved	`LIKELIHOOD:` High

Target

- programs/aol-staking-program/src/lib.rs#L731-L871

Description:

The program defines and configures an `early_unlock_min_seconds` parameter in `GlobalConfig`, exposes it in views, and documents it as a required waiting period before early unlock. However, the core early□unlock instruction `early_unlock_vault` does not enforce this minimum.

Relevant points:

- `early_unlock_min_seconds` is set in `initialize_global_config` and can be updated via `update_early_unlock_min`.
- `get_vault_info` uses `config.early_unlock_min_seconds` to compute `can_early_unlock`, so frontends and users see a boolean that appears to reflect on□chain rules.
- `early_unlock_vault` only checks time relative to `unlock_at` to calculate penalty tiers. It does not compare `Clock::get()?.unix_timestamp` to `locked_at + early_unlock_min_seconds` and has no error for "too early" unlock.
- Tests and docs (e.g. SECURITY_ANALYSIS, TESTING_GUIDE) assume that users "must wait early_unlock_min_seconds" before early unlock, and test names expect errors like `MinimumLockNotMet/EarlyUnlockTooEarly`, but the live logic does not enforce this.

Impact:

- If governance sets a non□zero `early_unlock_min_seconds` believing it to be a hard minimum lock period, users can still call `early_unlock_vault` immediately after staking, bypassing the intended time lock and only paying the configured penalty.

- This is a semantic and economic vulnerability: the protocol does not behave as documented or as UIs/reporting suggest, and time-based incentives/guarantees (e.g., marketing a “minimum 7-day lock”) are not actually enforced on-chain.

Recommendations:

It is recommended to consider the following mitigation measures:

- Enforce the minimum in `early_unlock_vault`:

Add a check before penalty calculation:

```
let config = &ctx.accounts.global_config;
if config.early_unlock_min_seconds > 0 {
    let min_early_unlock_time = vault
        .locked_at
        .checked_add(config.early_unlock_min_seconds)
        .ok_or(ErrorCode::MathOverflow)?;
    require!(
        current_time >= min_early_unlock_time,
        ErrorCode::MinimumLockNotMet
    );
}
```

- Introduce (or re-introduce) a dedicated error code such as `MinimumLockNotMet` with a clear message; ensure tests look for this exact error string.
- Align configuration and tests:
- Decide whether `early_unlock_min_seconds = 0` is allowed; if so, ensure `update_early_unlock_min` permits zero when resetting.
    - Update tests to cover:
        - Early unlock before the minimum (must fail with `MinimumLockNotMet`).
        - Early unlock after the minimum but before `unlock_at` (must succeed with penalty).

America.fun: Resolved with `@41694b830e ...`

Zenith: Verified.

[H-2] Old unfinalized snapshot can prematurely sweep rewards for newer snapshots

SEVERITY: High	IMPACT: High
STATUS: Resolved	LIKELIHOOD: Medium

Target

- programs/aol-staking-program/src/lib.rs#L587-L637

Description:

The `sweep_unclaimed_rewards` instruction allows any unswept (unfinalized) snapshot to be used as the timing gate for sweeping the entire `staking_bucket` balance to the treasury:

```
// (simplified version for report)
pub fn sweep_unclaimed_rewards(ctx: Context<SweepUnclaimedRewards>) ->
    Result<()> {
    let config = &ctx.accounts.global_config;
    let snapshot = &mut ctx.accounts.global_snapshot;
    let current_time = Clock::get()?.unix_timestamp;

    // PERMISSIONLESS
    let sweep_time = snapshot.snapshot_timestamp
        .checked_add(config.claim_window_seconds)
        .ok_or(ErrorCode::MathOverflow)?;

    require!(current_time >= sweep_time, ErrorCode::GracePeriodNotExpired);
    require!(!snapshot.is_finalized, ErrorCode::AlreadySwept);

    // Transfer remaining balance to treasury
    let remaining_balance = ctx.accounts.staking_bucket.amount;
    if remaining_balance > 0 {
        token::transfer(..., remaining_balance)?;
    }
    snapshot.is_finalized = true;
}
```

Because the amount swept is `staking_bucket.amount` at sweep time, not something tied to the specific snapshot (e.g. its own leftover), and because there is no check that the passed `global_snapshot` is the latest one, an attacker can:

1. Wait until there is an older snapshot `S0` whose claim window has expired but that has not been swept (`S0.is_finalized == false`).
2. Let a newer snapshot `S1` be taken and filled; users are still within `S1`'s claim window and validly expecting to claim.
3. Call `sweep_unclaimed_rewards` using `S0` as `global_snapshot`.
    - `current_time >= S0.timestamp + claim_window_seconds` is true.
    - `S0.is_finalized` is false.
    - The instruction transfers all current `staking_bucket.amount`, including rewards from `S1`, to the treasury and marks only `S0.is_finalized = true`.

This effectively steals/denies rewards from the current snapshot (and any later snapshots whose funds are still in the shared `staking_bucket`), even though their claim windows have not expired.

Recommendations:

It is recommended to bind sweeps to the correct snapshot and/or enforce ordering so old snapshots cannot be used to drain newer rewards.

America.fun: Resolved with @db54133718 ..., @2185d4e635 ..., @bfb8da5557 ... and @d4662c6bdb ...

Zenith: Verified.

4.3 Medium Risk

A total of 10 medium risk findings were identified.

[M-1] No partial user snapshot state: Users can permanently underclaim if they have to omit vaults

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Medium

Target

- programs/aol-staking-program/src/lib.rs#L458-L585

Description:

The `claim_rewards` flow for a given snapshot uses a per‑user `UserSnapshot` PDA keyed by `(user, snapshot_timestamp)` to store the user's total staking weight at that snapshot:

```
// In claim_rewards
let user_snapshot = &mut ctx.accounts.user_snapshot;
if (user_snapshot.user ≠ Pubkey::default()) {
    // Reuse recorded weight
    total_user_weight = user_snapshot.total_user_weight;
} else {
    // First time: compute from remaining_accounts
    ... compute total_user_weight from vaults ...
    user_snapshot.user = ctx.accounts.user.key();
    user_snapshot.snapshot_timestamp = snapshot.snapshot_timestamp;
    user_snapshot.total_user_weight = total_user_weight;
    user_snapshot.has_claimed = false;
    user_snapshot.amount_claimed = 0;
    user_snapshot.bump = ctx.bumps.user_snapshot;
}
require!(!user_snapshot.has_claimed, ErrorCode::AlreadyClaimed);
...
user_snapshot.has claimed = true;
```

Crucially:

- `UserSnapshot` is created/initialized in the same transaction as the actual claim, and `has_claimed` is set to `true` at the end of that transaction.
- Because Solana transactions are atomic, any failure in `claim_rewards` rolls back all writes (including `user_snapshot.total_user_weight` and `has_claimed`), so there is no durable "partial" user snapshot state from failed attempts.

However, this means that:

- On the first successful claim for a snapshot, the user's `total_user_weight` is whatever was computed from the `ctx.remaining_accounts` provided in that transaction.
- If the user (or their wallet/frontend) passes only a subset of their eligible vault accounts, the stored `total_user_weight` will be too small, and `user_share` will be under□computed. This could also be forced due to Solana's transaction size limit.
- After that claim, `user_snapshot.has_claimed` is `true`, and any subsequent attempt to correct the mistake will hit `require!(!user_snapshot.has_claimed, ...)` and fail. The user has permanently "locked in" a lower weight and cannot claim the remainder; the unclaimed portion of rewards will eventually be swept to the treasury.

This is a user□harmful behavior that can be triggered by misconfigured or malicious front□ends (e.g., a UI that only passes some of the user's vaults to `claim_rewards`), causing users to forfeit part of their legitimate rewards without any way to remedy it.

In addition, the implicit vault limit per transaction on Solana makes this underclaim behavior worse: `claim_rewards` requires callers to pass all their vaults as `remaining_accounts`, but there is no on-chain loop bound except Solana's transaction size and compute limits. In practice, a user with many vaults can only fit around 24 vault accounts into a single `claim_rewards` transaction, which contradicts `ABSOLUTE_MAX_VAULTS_PER_USER` (40); beyond that, the transaction will fail at the runtime level. Because the program only allows one claim per snapshot and freezes `UserSnapshot.total_user_weight` on that first successful claim, any user with more eligible vaults than can fit in one transaction is structurally forced to claim based on a subset of their vaults and permanently underclaim the rest of their rewards for that snapshot.

Recommendations:

It is recommended to either enforce completeness of the first claim or redesign the claim mechanism to support safe partial claims.

America.fun: Resolved with `@5b48c1d63c ...`

Zenith: Verified.

[M-2] `withdraw_vault`, `early_unlock_vault`, and `admin_force_unstake` can be DoS by donation attack

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Medium

Target

- lib.rs#L687
- lib.rs#L825
- lib.rs#L2234

Description:

`withdraw_vault`, `early_unlock_vault`, and `admin_force_unstake` are intended to withdraw funds and close `vault_aol_account` to reclaim rent and finalize the vault lifecycle.

However, they assume `vault_aol_account` can always be closed, but an attacker can DoS the flow by donating tokens to `vault_aol_account`, causing `token::close_account()` to fail due to a non-zero balance.

This can permanently block withdrawals/early unlocks/forced unstaking, leaving funds stuck and preventing expected protocol operations.

Recommendations:

Consider transferring out any excess balance in `vault_aol_account` to the user.

America.Fun: Resolved with `@0c647f97d9 ...`

Zenith: Verified.

[M-3] Remove `admin_close_all_configs` and `admin_empty_protocol_accounts` in production code

`SEVERITY:` Medium	`IMPACT:` High
`STATUS:` Resolved	`LIKELIHOOD:` Low

Target

- lib.rs#L1285-L1442

Description:

`admin_close_all_configs` and `admin_empty_protocol_accounts` are “testing cleanup” functions to reset/cleanup protocol state for local testing and repeated deployments.

They should be removed from the production code to prevent any accidental or malicious clean-up of the on-chain states.

Recommendations:

Remove these functions from mainnet builds.

America.Fun: Resolved with `@e53afecccd ...`

Zenith: Verified. Resolved by removing the test cleanup functions.

[M-4] Incorrect close pattern in `admin_force_unstake` causes runtime failure

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Medium

Target

- programs/aol-staking-program/src/lib.rs#L2234

Description:

The `AdminForceUnstake` context uses Anchor's `close = admin` attribute on a SPL token account:

```
#[account(
    mut,
    seeds = [b"vault_aol", vault.key().as_ref()],
    bump,
    close = admin
)]
pub vault_aol_account: Account<'info, TokenAccount>,
```

However, `vault_aol_account` is owned by the SPL Token program, not by your program. Anchor's `close = admin` mechanism tries to drain lamports from that account at the end of the instruction via the System Program, which leads to the runtime error:

```
instruction spent from the balance of an account it does not own
```

Even though the core logic uses a correct PDA authority for the SPL `transfer`, the auto-close step is invalid for a token account that your program does not own. This breaks `admin_force_unstake` in practice.

Recommendations:

It is recommended to remove `close = admin` from the SPL token account and close it explicitly via the token program with the PDA authority:

- In `AdminForceUnstake`, keep `close = admin` on the `vault` (your own account), but remove `close = admin` from `vault_aol_account`:

```
#[account(
    mut,
    seeds = [b"vault_aol", vault.key().as_ref()],
    bump
)]
pub vault_aol_account: Account<'info, TokenAccount>,
```

- In `admin_force_unstake` handler, after transferring `amount_staked` back to the user, explicitly close the token account using SPL `close_account` and the PDA signer:

```
token::close_account(
    CpiContext::new_with_signer(
        ctx.accounts.token_program.to_account_info(),
        CloseAccount {
            account: ctx.accounts.vault_aol_account.to_account_info(),
            destination: ctx.accounts.admin.to_account_info(), // or user,
    as desired
            authority: ctx.accounts.vault_aol_account.to_account_info(),
        },
        signer_seeds,
    ),
)?;
```

America.fun: Resolved with `@089cef9e91 ...` and `@ae80043716 ...`

Zenith: Verified.

[M-5] `create_vault` will be DoS for user after multiple creations due to overflow in `total_vaults_created`

`SEVERITY:` Medium	`IMPACT:` Medium
`STATUS:` Resolved	`LIKELIHOOD:` Medium

Target

- lib.rs#L275-L277

Description:

`VaultRegistry.total_vaults_created` is intended to track how many vaults a user has created while enforcing the active-vault cap (e.g., 40).

However, it is stored as a `u8`, so repeated create/close/recreate cycles eventually overflow; after 255 creations, `checked_add(1)` fails even if the user has fewer than 40 active vaults.

This permanently prevents the user from creating new vaults, creating an avoidable long-term DoS failure unrelated to the active-vault limit.

Recommendations:

Store `total_vaults_created` in a wider type (e.g., `u64`).

America.Fun: Resolved with `@ab6fe5020a ...`

Zenith: Verified. Resolved by increasing `total_vaults_created` to `u32`.

[M-6] Events emit stale token balances after CPIs

SEVERITY: Medium	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: High

Target

- aol-staking-program/src/lib.rs#L1149
- programs/aol-staking-program/src/lib.rs#L1189

Description:

Two instructions emit events from `TokenAccount.amount` immediately after performing SPL Token CPI transfers, but without reloading the token account. In Anchor, `ctx.accounts.<token_account>.amount` holds the value from when the account was first deserialized, so after a CPI transfer the in-memory struct is stale:

- `admin_fill_intermediary`:

```
  token::transfer(...)?;
  emit!(IntermediaryFilledEvent {
    amount,
    new_balance: ctx.accounts.intermediary_bucket.amount,
    timestamp: Clock::get()?.unix_timestamp,
});
```

`new_balance` still reflects the pre-transfer amount.

- `admin_withdraw_intermediary`:

```
  token::transfer(...)?;
  emit!(IntermediaryWithdrawnEvent {
    amount,
    remaining_balance: ctx.accounts.intermediary_bucket.amount,
    timestamp: Clock::get()?.unix_timestamp,
});
```

`remaining_balance` is likewise stale.

While this does not affect actual balances (the token program state is correct), it can mislead off-chain consumers (indexers, dashboards) that rely on event fields or view outputs for accurate balances.

Recommendations:

It is recommended to reload token accounts after CPIs and before emitting events:

- In any instruction that:
    - Calls `token::transfer` / `token::mint_to` / `token::burn` (or similar), and
    - Then emits an event or returns a view that includes `.amount` from the same account,

  insert:

```
ctx.accounts.<token_account>.reload()?;
```

before reading `.amount` for events or return structs.

- Concretely:
    - In `admin_fill_intermediary`, call `ctx.accounts.intermediary_bucket.reload()?;` before using `intermediary_bucket.amount` in `IntermediaryFilledEvent`.
    - In `admin_withdraw_intermediary`, reload `intermediary_bucket` before emitting `IntermediaryWithdrawnEvent`.

America.fun: Resolved with `@2b0bee3179 ...`

Zenith: Verified.

[M-7] Timing invariant (snapshot vs. claim window) is not enforced

SEVERITY: Medium	IMPACT: High
STATUS: Resolved	LIKELIHOOD: Low

Target

- programs/aol-staking-program/src/lib.rs#L28-L30

Description:

The protocol's timing section documents a critical invariant between `snapshot_interval_seconds` and `claim_window_seconds`:

```
// □  CRITICAL: snapshot_interval MUST be > claim_window to allow sweeps
    before next snapshot!
```

However, this relationship is only enforced in comments and default constants, not on-chain:

- `initialize_global_config` sets both fields from constants but performs no relational check.
- `update_claim_window` and `update_snapshot_interval` only enforce `new_value > 0`:
- There is no check that `snapshot_interval_seconds > claim_window_seconds` (plus any sweep grace) after admin updates.

As a result, an admin (or compromised admin key) can misconfigure:

- `snapshot_interval_seconds ≤ claim_window_seconds`, or
- Values where the sweep cannot reasonably happen between windows,

leading to:

- Snapshots taken while the previous claim window is still open.
- "Leftover" rewards from one snapshot bleeding into the next in ways that contradict the intended model and the written docs.

This is a consistency and economic behavior bug: the protocol can enter timing states that contradict its own invariant, and this is not prevented on-chain.

Recommendations:

It is recommended to enforce the documented timing invariant in the update paths:

- In `update_claim_window`, require that the resulting configuration still satisfies the invariant, e.g.:

```
pub fn update_claim_window(ctx: Context<AdminOnly>, new_value: i64) ->
    Result<()> {
    require!(new_value > 0, ErrorCode::InvalidParameter);
    let snapshot_interval
    = ctx.accounts.global_config.snapshot_interval_seconds;
    require!(
        snapshot_interval > new_value,
        ErrorCode::InvalidParameter
    );
    // existing update + event
}
```

- In `update_snapshot_interval`, similarly ensure:

```
pub fn update_snapshot_interval(ctx: Context<AdminOnly>, new_value: i64) ->
    Result<()> {
    require!(new_value > 0, ErrorCode::InvalidParameter);
    let claim_window = ctx.accounts.global_config.claim_window_seconds;
    require!(
        new_value > claim_window,
        ErrorCode::InvalidParameter
    );
    // existing update + event
}
```

- If the intended invariant also involves `sweep_grace_period_seconds`, extend the checks to:

```
snapshot_interval_seconds >= claim_window_seconds + sweep_grace_period_seconds
```

America.fun: Resolved with `@1deb2c1aa7 ...` and `@74ba84b608 ...`

Zenith: Verified.

[M-8] `execute_daily_snapshot` fails to prevent negative/old/future `snapshot_timestamp`

SEVERITY: Medium	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: High

Target

- lib.rs#L356-L360

Description:

`execute_daily_snapshot` is intended to create the protocol's daily snapshot for reward accounting.

However, it does not reject a `snapshot_timestamp` that is negative, in the past or future, allowing snapshots to be created outside the valid time range.

If `snapshot_timestamp` is in the future, vaults created after snapshot execution but before that timestamp can become eligible even though `total_global_weight` used in the snapshot was captured before those vaults existed. The same issue applies when `snapshot_timestamp` is in the past. This can break snapshot ordering and reward calculations, enabling unfair reward distribution.

If `snapshot_timestamp` is negative such that `snapshot_timestamp - current_time == i64::MIN`, then `(snapshot_timestamp - current_time).abs()` will wrap and stay negative due to this issue, making `time_diff` a negative number and pass the `time_diff <= 60` check. Though this will not pass the next `time_since_last >= config.snapshot_interval_seconds` check, it is recommended not to allow negative `snapshot_timestamp`.

Recommendations:

Derive `snapshot_timestamp` from `current_time`.

America.Fun: Resolved with `@3ba5f4b5ad ...`

Zenith: Verified. Resolved by using `current_time` as snapshot timestamp.

[M-9] `get_user_portfolio` and `get_user_claimable` cannot handle `ABSOLUTE_MAX_VAULTS_PER_USER` due to Solana message size limit

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Medium

Target

- programs/aol-staking-program/src/lib.rs#L1522-L1566

Description:

The protocol allows up to `ABSOLUTE_MAX_VAULTS_PER_USER = 40`, and both view instructions rely on `ctx.remaining_accounts` to receive all of a user's vaults. Even though these are "view" methods, Anchor implements them via transaction simulation; a full Solana message (with all remaining accounts) is constructed and sent to `simulateTransaction`, so the 1232byte message size limit still applies.

Both `get_user_portfolio` and `get_user_claimable` are designed to aggregate across "all vaults passed in `remaining_accounts`", but:

- `get_user_portfolio` can handle ≤ 31 vaults.
    - `get_user_claimable` can handle ≤ 30 vaults.

For users at the configured maximum (40 vaults), a single view call cannot see all vaults; callers must either:

- Paginate manually and stitch results off-chain, or
    - Risk incomplete / misleading information if they assume a single call is exhaustive.

Recommendations:

It is recommended to reconcile vault-count configuration and view API design with Solana's message size limit.

America.fun: Resolved with `@4b469fca95 ...` and `@8a3dfc5153 ...`

Zenith: Verified. Resolved by reducing `ABSOLUTE_MAX_VAULTS_PER_USER` to 25 and by performing further checks.

[M-10] Hardcoded Rapid Testing should be removed for production

SEVERITY: Medium	IMPACT: High
STATUS: Resolved	LIKELIHOOD: Low

Target

- lib.rs#L32-L41
- lib.rs#L123-L147
- lib.rs#L239-L245
- lib.rs#L335-L337

Description:

The program is hardcoded to use “mainnet rapid testing” values that accelerate time-based logic, which is inappropriate for production and can desynchronize snapshot/claim accounting.

This can cause incorrect reward distribution when deployed to mainnet.

Recommendations:

Remove hardcoded rapid-testing parameters and load production timing config instead.

America.Fun: Resolved with `@9e66a972fd ...` and `@4b469fca95 ...`

Zenith: Verified.

4.4 Low Risk

A total of 15 low risk findings were identified.

[L-1] Generic token account constraints allows caller to send penalty and swept rewards to arbitrary accounts

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- lib.rs#L2078-L2083
- lib.rs#L1976-L1981

Description:

`early_unlock_vault` is intended to let users unlock early while enforcing an economic penalty by transferring the penalty fee to the protocol's penalty wallet account.

However, `penalties_aol_account` is accepted as a generic token account rather than enforcing the penalty wallet's canonical associated token account.

This allows users to initialize an arbitrary token account with owner set to penalty wallet, and pass it under `penalties_aol_account` for `early_unlock_vault`.

That means the penalty fee can be sent to an account with a different address from the expected account. Though the penalty wallet is the account's owner, this creates inconvenience as the administrator now has to consolidate the penalty fees from multiple accounts.

The same issue also applies for `sweep_unclaimed_rewards`, as it allows one to pass in a generic token account for treasury as well.

Recommendations:

Require `penalties_aol_account` and `treasury_usd1_account` to be the associated token account for the expected mint.

America.Fun: Resolved with `@3acea9f366 ...` and `@008e1e8cbb ...`

Zenith: Verified.

[L-2] Lack of validation in `update_lock_tiers` and `update_penalty_tiers`

SEVERITY: Low	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Low

Target

- lib.rs#L977
- lib.rs#L995

Description:

`update_lock_tiers` and `update_penalty_tiers` are intended to update tiered configuration used to determine lock durations and early-unlock penalties. However, they do not validate that tier entries are unique and that `PenaltyTier` thresholds are strictly ordered (expected descending), allowing ambiguous or conflicting tier resolution.

This can cause incorrect tier selection and penalty calculation, leading to mispriced unlocks.

Recommendations:

When updating tiers, enforce uniqueness (no duplicate thresholds/durations) and require strict ordering (e.g., `PenaltyTier` in descending order with monotonic ranges), rejecting any configuration that violates these invariants.

America.Fun: Resolved with `@dd3e5e7652 ...`

Zenith: Verified. Resolved with additions of `validate_penalty_tiers()` and `validate_lock_tiers()`.

[L-3] Misleading “Grace Period” error and timing semantics around claim and sweep

SEVERITY: Low	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Low

Target

- programs/aol-staking-program/src/lib.rs#L470
- programs/aol-staking-program/src/lib.rs#L600

Description:

The claim and sweep timing checks use the same window (`claim_window_seconds`) but expose two different error names that suggest distinct phases:

```
// In claim_rewards
require!(current_time <= claim_deadline, ErrorCode::ClaimWindowExpired);

// In sweep_unclaimed_rewards
require!(current_time >= sweep_time, ErrorCode::GracePeriodNotExpired);
```

Both `claim_deadline` and `sweep_time` are computed as:

```
claim_deadline / sweep_time = snapshot.snapshot_timestamp
    + config.claim_window_seconds;
```

So:

- A claim succeeds while `current_time ≤ snapshot_timestamp + claim_window_seconds`.
- A sweep is allowed as soon as `current_time ≥ snapshot_timestamp + claim_window_seconds`.

Potentially, a sweep and claim can happen at the same time when `current_time == snapshot_timestamp + claim_window_seconds`. Furthermore, there is no additional grace period beyond the claim window, yet the error is named `GracePeriodNotExpired` and the config includes a `sweep_grace_period_seconds` field that is never used. This makes it look like there is a separate “claim window” followed by a distinct “grace period” before sweeping, when in fact they are the same instant.

Recommendations:

It is recommended to align naming and logic so timing behavior is explicit and unambiguous.

America.fun: Resolved with `@df7769f04d ...`

Zenith: Verified.

[L-4] Loose validation of `user_snapshot` account in `get_user_claimable` instruction

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Medium

Target

- programs/aol-staking-program/src/lib.rs#L1899-L1900

Description:

In the `GetUserClaimable` context, `user_snapshot` is declared as a `/// CHECK`ed optional account:

```
#[derive(Accounts)]
pub struct GetUserClaimable<'info> {
    // ...
    /// CHECK: Optional account - may not exist if user hasn't claimed yet
    pub user_snapshot: Option<UncheckedAccount<'info>>,
}
```

Inside `get_user_claimable`, this account is read and Borsh-deserialized without fully verifying that it is actually a `UserSnapshot` owned by this program:

```
if let Some(user_snap_acc) = &ctx.accounts.user_snapshot {
    if !user_snap_acc.data_is_empty() {
        let data = user_snap_acc.try_borrow_data()?;
        if data.len() > 8 {
            use borsh::BorshDeserialize;
            let data_slice: &[u8] = &data[8..]; // Skip 8-byte discriminator
            match UserSnapshot::try_from_slice(data_slice) {
                Ok(user_snap) => {
                    total_user_weight_at_snapshot
    = user_snap.total_user_weight;
                    has_claimed = user_snap.has_claimed;
                    // ...
                }
                // ...
            }
            }
        }
    }
```

There is no explicit check that:

- `user_snap_acc.owner == program_id`, or
- The first 8 bytes of `data` match `UserSnapshot`'s discriminator.

Because `get_user_claimable` is a view-only instruction that does not move tokens or modify state, this does not lead to a direct exploit (a malicious account can at most cause misleading "claimable" numbers). However, the `/// CHECK` comment can give the impression that ownership/type are being validated when they are not.

Recommendations:

It is recommended to tighten validation of the `user_snapshot` CHECK account:

- Before deserializing, add explicit checks such as:

```
require!(user_snap_acc.owner == &crate::ID,
    ErrorCode::UnauthorizedVaultAccess);
let data = user_snap_acc.try_borrow_data()?;
require!(data.len() >= 8, ErrorCode::InvalidVaultAccount);
require!(
    &data[0..8] == &UserSnapshot::discriminator(),
    ErrorCode::InvalidVaultAccount
);
```

- Then slice off the discriminator and call `UserSnapshot::try_from_slice` as you already do.

America.fun: Resolved with @f28c5d1b6e ...

Zenith: Verified.

[L-5] Unused limit constants and errors give a false sense of protection

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Medium

Target

- programs/aol-staking-program/src/lib.rs#L56-L57
- programs/aol-staking-program/src/lib.rs#L2943-L2954

Description:

The staking program defines several limit-related constants and error codes that are never actually used in instruction logic:

- `MAX_VAULTS_PER_SNAPSHOT`, `MAX_VAULTS_PER_CLAIM` are declared but never referenced.
- `ErrorCode::TooManyVaults` and `ErrorCode::PreviousSnapshotNotSwept` are defined but never returned by any `require!` or `err!`.

At the same time, docs and testing guides refer to these as if they enforce per-transaction vault limits or require previous snapshots to be swept before new ones. In practice, snapshot and claim instructions simply iterate over `ctx.remaining_accounts` up to runtime limits, and new snapshots are allowed even when the previous snapshot's rewards haven't been swept.

Recommendations:

It is recommended to either wire these constants and errors into the on-chain checks (e.g., enforce max vaults per snapshot/claim and optionally block new snapshots while old ones are unswept), or remove the unused items and adjust documentation/tests so that the implemented behavior and the documented guarantees match, avoiding any mistaken assumptions about protections that do not actually exist.

America.fun: Resolved with `@11c97f3fe6 ...`

Zenith: Verified.

[L-6] `StakingVault.total_claimed` and `last_claim_at` are never updated leading to incorrect `get_vault_info.total_claimed`

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Medium

Target

- programs/aol-staking-program/src/lib.rs#L2438-L2439

Description:

The `StakingVault` struct defines `last_claim_at` and `total_claimed` and these fields are initialized to `0` in `create_vault`. However, they are never updated when rewards are claimed (`claim_rewards`, `get_user_claimable`), nor anywhere else in the codebase. Despite this, `get_vault_info` exposes `total_claimed`.

As a result, `get_vault_info.total_claimed` is always `0` from the chain's perspective, even if the user has claimed rewards, and `last_claim_at` is always `0`. This does not affect the correctness of reward distribution (which is snapshot-based and uses `UserSnapshot`/`GlobalSnapshot`), but it is misleading for frontends or analytics that rely on per-vault claim history.

Recommendations:

It is recommended to either wire these fields into the reward flow or remove them to avoid confusion.

America.fun: Resolved with `@3098612b9b ...` and `@bfb8da5557 ...`

Zenith: Verified. Resolved by integrating these fields into the reward flow.

[L-7] `early_unlock_min_seconds` inconsistency and one□way behavior

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Medium

Target

- programs/aol-staking-program/src/lib.rs#L42

Description:

The configuration field `early_unlock_min_seconds` in `GlobalConfig` is currently inconsistent and effectively one□way:

- `initialize_global_config` sets `early_unlock_min_seconds` from `DEFAULT_EARLY_UNLOCK_MIN`, which is currently `0`.
  - `update_early_unlock_min` requires `new_value > 0`:
  - This means:
    - The protocol can start with `early_unlock_min_seconds = 0`.
    - Once any positive minimum is set, it can never be reset back to 0 via the admin API, even though 0 is a valid initial value.

This combination leads to surprising admin UX (irreversible config direction) and inconsistent expectations.

Recommendations:

It is recommended to clarify and decide the intended semantics:

- If a “no minimum wait” mode is valid:
  - Treat `0` as a first□class, reversible value.
- If a minimum wait must always be strictly positive:
  - Enforce that invariant at initialization as well, not just in `update_early_unlock_min`.

America.fun: Resolved with `@d0873149e3 ...`

Zenith: Verified. Resolved by allowing `0` as valid value.

[L-8] `get_user_portfolio` and `get_user_claimable` can multi-count vaults via duplicate remaining accounts

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Medium

Target

- programs/aol-staking-program/src/lib.rs#L1522-L1566

Description:

Both view instructions derive a user's total weight and stake by iterating over `ctx.remaining_accounts` and summing per□vault values without any deduplication:

- `get_user_portfolio`
- `get_user_claimable` (when no `user_snapshot` is used)

On Solana, an instruction can include the same account multiple times in its account list. Anchor exposes this as repeated entries in `ctx.remaining_accounts`. Since both methods trust `remaining_accounts` and only validate each account with `validate_vault_account` (owner + seeds) but never track which vaults they have already processed, a malicious or buggy client can:

- Include the same vault PDA many times in `remaining_accounts`.
- Cause `get_user_portfolio` to:
  - Push duplicate `VaultSummary` entries for the same vault.
  - Inflate `total_staked` and `total_weight` by counting the same vault repeatedly.
- Cause `get_user_claimable` (when deriving weight from vaults) to:
  - Inflate `total_user_weight_at_snapshot`, and thus the reported `claimable_amount`, by counting the same vault's weight multiple times.

These are view-only instructions, so they do not directly move funds, but they corrupt the on-chain—computed views. Any off-chain logic, UI, or monitoring that trusts these values (e.g. dashboards, risk metrics, "what can I claim?" previews) can be significantly misled.

Recommendations:

It is recommended to treat `remaining_accounts` as untrusted and deduplicate vaults before aggregation.

America.fun: Resolved with `@8a3dfc5153 ...` and `@0d65335297 ...`

Zenith: Verified.

[L-9] Early-withdrawal with penalty still allowed after vault unlock time

`SEVERITY:` Low	`IMPACT:` Medium
`STATUS:` Resolved	`LIKELIHOOD:` Low

Target

- programs/aol-staking-program/src/lib.rs#L732-L768

Description:

The protocol exposes two ways to exit a vault:

- `withdraw_vault`: free withdrawal, gated by `current_time ≥ vault.unlock_at`
- `early_unlock_vault`: penalized early withdrawal, but does not check that `current_time < vault.unlock_at`:

As a result, once `current_time ≥ vault.unlock_at` (i.e. during the normal, penalty-free withdrawal window), users can still call `early_unlock_vault` and unnecessarily pay a penalty, even though `withdraw_vault` would let them withdraw the full amount. This is not a direct theft vector (the user is harming themselves), but it is a sharp edge and UX/economic bug: an interface bug or misrouted call can cause users to lose funds via a penalty that should no longer apply after the lock has matured.

Recommendations:

It is recommended to disable the penalty path once the regular withdrawal window is open, so users cannot be charged a penalty after `unlock_at`.

America.fun: Resolved with `@fa5695faf9 ...`

Zenith: Verified.

[L-10] `emit!()` and `msg!()` can be truncated by RPC due to log limits

SEVERITY: Low	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- lib.rs

Description:

The program uses `emit!()` and `msg!()` in several parts of the code to emit log/events.

However, these logs can be truncated or dropped by RPC due to log limits, especially in large transactions or loops, making critical events unreliable to observe off-chain.

Recommendations:

Use `emit_cpi!()` instead for critical events and minimize use of `msg!()` and `emit!()`.

America.Fun: Acknowledged. RPC log truncation is a Solana/runtime behavior, not something the program can fully prevent. `emit!()` already writes an on-chain event log, but it's still delivered through the same runtime log stream that RPC can truncate. `emit_cpi!()` is only useful if you need another program to consume the event via CPI, and it changes integration/IDL expectations; it's not a drop-in "make logs reliable" switch.

[L-11] Unrealistic `MAX_VAULTS_PER_SNAPSHOT` / `MAX_VAULTS_PER_CLAIM` constants vs Solana transaction limits

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Medium

Target

- programs/aol-staking-program/src/lib.rs#L55-L57

Description:

The program defines the following constants:

```
// Maximum vaults that can be processed in a single transaction
pub const MAX_VAULTS_PER_SNAPSHOT: usize = 500;
pub const MAX_VAULTS_PER_CLAIM: usize = 100;
```

These values are labeled as "maximum vaults that can be processed in a single transaction", but they do not align with Solana's practical transaction limits. In reality, due to the 1232-byte message size cap and account/compute limits, a single transaction can only include dozens of vault accounts (typically on the order of 20—40), not 100—500. Moreover, these constants are not actually enforced anywhere in the code; neither `execute_daily_snapshot` nor `claim_rewards` checks `ctx.remaining_accounts.len()` against them. This combination (unrealistic values + no enforcement) can mislead readers into believing that the program both supports and enforces much higher per□transaction vault counts than Solana can handle.

Recommendations:

It is recommended to either enforce realistic limits based on Solana's constraints or remove these constants to avoid confusion.

America.fun: Resolved with `@11c97f3fe6 ...`

Zenith: Verified. Resolved by removing.

[L-12] Using `msg()!` in for loop within `claim_rewards()` could use excessive CU and fail

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Medium

Target

- lib.rs#L488-L504

Description:

The for loop in `claim_rewards()`calls `msg!()` within itself, which could significantly increases compute usage per iteration and scales linearly with number of vault accounts.

This can cause transactions to exceed the compute unit limit and prevent claiming of rewards for multiple vaults accounts.

Recommendations:

Remove `msg!()` to keep CU minimal.

America.Fun: Resolved with `@10d1c9f7c2 ...`

Zenith: Verified. Resolved by removing the `msg!()` calls in the for loops.

[L-13] Incorrect `intermediary_transferred` emitted in `SnapshotExecutedEvent`

`SEVERITY:` Low	`IMPACT:` Low
`STATUS:` Resolved	`LIKELIHOOD:` Low

Target

- lib.rs#L447-L451

Description:

`SnapshotExecutedEvent` is intended to emit an accurate record of how much reward funding was actually transferred during a snapshot execution.

However, it reports `config.daily_fill_amount` instead of the real `transfer_amount` used in the transfer.

This can cause incorrect tracking/display of the transferred amount.

Recommendations:

Emit `transfer_amount` in `SnapshotExecutedEvent` instead.

America.Fun: Resolved with `@90e2e9ecf8 ...`

Zenith: Verified. Resolved by emitting `intermediary_transferred`, which is set to `transfer_amount`.

[L-14] `sweep_unclaimed_rewards` should check for `current_time > sweep_time` instead

`SEVERITY:` Low	`IMPACT:` Medium
`STATUS:` Resolved	`LIKELIHOOD:` Low

Target

- lib.rs#L600

Description:

`sweep_unclaimed_rewards` is intended to move unclaimed rewards only after the snapshot's claim period has fully ended.

However, it allows sweeping when `current_time >= sweep_time`, even though rewards are still claimable at `current_time == sweep_time`.

This creates an off-by-one boundary where users can be front-run by the sweep at the exact cutoff timestamp, causing legitimate claims to fail and rewards to be incorrectly swept.

Recommendations:

Change the condition to `require!(current_time > sweep_time, ...)` so that sweeping is only possible strictly after the final claimable second.

America.Fun: Resolved with `@df7769f04dc ...`

Zenith: Verified. Resolved by adding a grace period after claim deadline, to ensure sweeping only possible after claim deadline.

[L-15] `claim_rewards` should validate `!snapshot.is_finalized`

SEVERITY: Low	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Low

Target

- lib.rs#L458

Description:

`claim_rewards` is intended to let users claim rewards from an active snapshot within its configured claim window.

However, it does not enforce `!snapshot.is_finalized`, so if `claim_window_seconds` is later increased, users may be able to claim against snapshots that were already finalized and swept.

Recommendations:

Add a hard guard `require!(!snapshot.is_finalized, ...)` in `claim_rewards` so finalized/swept snapshots can never be claimed, regardless of later timing config changes.

America.Fun: Resolved with `@ddd1894c91 ...`

Zenith: Verified. Resolved by checking snapshot is not finalized during `claim_rewards`.

4.5 Informational

A total of 10 informational findings were identified.

[I-1] Comprehensive test suite is out-of-sync with program behavior and not reliable as shipped

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- tests/aol-staking-program.ts

Description:

The `tests/aol-staking-program.ts` suite is advertised as a "comprehensive" test harness with full function coverage. In practice:

- Many tests fail even when the program and environment are configured exactly as documented for local validator testing.
- Several tests expect behaviors that the on-chain program does not implement (e.g., enforced `early_unlock_min_seconds`, specific timing invariants between snapshot and claim window, particular snapshot argument signatures).
- Some admin/tier configuration tests rely on legacy or changed semantics that no longer match `lib.rs`.

As a result, the test suite is not "ready" in its current form: it cannot be treated as a definitive indicator that the deployed program matches its intended design or security properties.

Recommendations:

It is recommended to bring the test suite back into alignment with the program by revisiting each failing test and deciding, per case, whether:

- The program should be updated to match the intended behavior (e.g., early-unlock minimum enforcement, timing invariants), or
    - The test should be updated or removed because it reflects deprecated assumptions.

America.fun: Resolved with `@a3df68a88b ...`

Zenith: Verified. Resolved by removing.

[I-2] Hardcoded test success summary printed even when tests fail

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- tests/aol-staking-program.ts#L2036-L2061

Description:

The comprehensive test suite prints a hardcoded “all tests passed” summary regardless of actual test results:

```
after("Test Summary", () ⇒ {
  console.log("\n" + "=".repeat(60));
  console.log("□ ALL COMPREHENSIVE TESTS COMPLETED SUCCESSFULLY!");
  console.log("=".repeat(60));
  console.log("\nTest Coverage:");
  console.log("  □ Phase 2: Initialization (4 tests)");
  console.log("  □ Phase 3: Staking (5 tests)");
  console.log("  □ Phase 4: Snapshot & Rewards (6 tests)");
  console.log("  □ Phase 5: Withdrawals (2 tests)");
  console.log("  □ Phase 6: Admin Updates (8 tests)");
  console.log("  □ Phase 7: Admin Operations (1 test)");
  console.log("  □ Phase 8: View/Query Functions (4 tests)");
  console.log("  □ Phase 9: Additional Admin Config (5 tests)");
  console.log("  □ Phase 10: Tier Management (2 tests)");
  console.log("  □ Phase 11: Additional View Functions (1 test)");
  console.log("  □ Phase 12: Security & Access Control (4 tests)");
  console.log("  □ Phase 13: Boundary Conditions (6 tests)");
  console.log("  □ Phase 14: Data Integrity (3 tests)");
  console.log("\n  □ TOTAL: 51 comprehensive tests");
  console.log("  □ Function Coverage: 100% (33/33 functions)");
  console.log("\n" + "=".repeat(60) + "\n");
});
```

Even when multiple tests fail, this `after` hook still prints a full green “ALL COMPREHENSIVE TESTS COMPLETED SUCCESSFULLY!” summary. This is misleading for developers and auditors, especially in CI or when scanning logs quickly, because it contradicts

Mocha/Anchor's actual failure output.

Recommendations:

It is recommended to remove or conditionally gate the hardcoded success summary so it reflects the real test outcome.

America.fun: Resolved with `@a3df68a88b ...`

Zenith: Verified. Resolved by removing.

[I-3] Unused `GlobalConfig.snapshot_counter` field

`SEVERITY:` Informational	`IMPACT:` Informational
`STATUS:` Resolved	`LIKELIHOOD:` Low

Target

- programs/aol-staking-program/src/lib.rs#L2411

Description:

The `GlobalConfig` struct defines a `snapshot_counter` field intended as a "Counter for snapshot PDAs".

However, this field is only initialized to `0` in `initialize_global_config` and is never incremented, read, or used anywhere else in the program. Snapshot PDAs are currently derived solely from `snapshot_timestamp`:

As a result, `snapshot_counter` is dead state: it doesn't affect behavior, and any reader assuming snapshots are indexed or constrained by this counter would be mistaken. There is no direct vulnerability, but it increases confusion about how snapshots are identified and managed.

Recommendations:

It is recommended to either use `snapshot_counter` if intended or remove it to avoid confusion.

America.fun: Resolved with `@db54133718 ...`

Zenith: Verified. Snapshot counter is now used.

[I-4] Redundant `staking_bucket` account in `get_user_claimable` context

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- programs/aol-staking-program/src/lib.rs#L1896-L1897

Description:

The `GetUserClaimable` accounts context includes a fully constrained `staking_bucket` `TokenAccount` PDA. However, the implementation of `get_user_claimable` never reads from `ctx.accounts.staking_bucket`.

This adds an extra required account and compute budget without contributing to the computation, and may confuse readers into thinking staking-bucket balance is part of the claimable calculation when it is not (the function uses `snapshot.initial_staking_balance` instead).

Recommendations:

It is recommended to either use `staking_bucket` if intended or remove it from the context.

America.fun: Resolved with `@31a2d650f1 ...`

Zenith: Verified. Resolved by removing.

[I-5] Centralization risk: Admin can unilaterally drain all reward buckets

SEVERITY: Informational	IMPACT: Informational
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- programs/aol-staking-program/src/lib.rs

Description:

The program grants the configured `admin` full, unilateral control over all USD1 reward buckets and accumulated rewards:

- `admin_withdraw_intermediary` allows the admin to transfer any amount of USD1 from the `intermediary_bucket` PDA to an arbitrary admin-owned token account.
- `admin_empty_protocol_accounts` is explicitly designed to drain all USD1 from both `intermediary_bucket` and `staking_bucket` into an admin-controlled account, and is gated only by `admin`.

While this is intentionally documented as a “testing/cleanup/emergency” mechanism, on any live deployment this means the admin can, at any time, seize all unclaimed and future rewards, regardless of users’ expectations. This is a centralization / trust-risk rather than a correctness bug.

Recommendations:

It is recommended to make this centralization risk explicit and, if desired, mitigate it via on-chain controls:

- At minimum, transparency:
  - Clearly document in on-chain comments and external docs that:
    - `admin_withdraw_intermediary` and `admin_empty_protocol_accounts` give the admin full custody over protocol reward funds.
    - Users are trusting the admin not to arbitrarily drain rewards.
  - Consider exposing these capabilities clearly in the security analysis / risk disclosures.
- Optional hardening (depending on desired trust model):
  - Introduce a separate “emergency” or “treasury” authority for draining functions,

distinct from the day-to-day admin used for configuration.
- Add a time-lock or multi-step process (e.g., propose+execute with delay) for `admin_empty_protocol_accounts`, giving off-chain governance or users time to react.

America.fun: Acknowledged.

[I-6] Legacy sweep grace-period setting is unused code

`SEVERITY`: Informational	`IMPACT`: Informational
`STATUS`: Resolved	`LIKELIHOOD`: Low

Target

- programs/aol-staking-program/src/lib.rs#L35

Description:

The config field `sweep_grace_period_seconds` and its default `DEFAULT_GRACE_PERIOD` are explicitly marked as “legacy — not used for sweep timing” in the code, and the sweep logic (`sweep_unclaimed_rewards`) uses only `claim_window_seconds` to gate sweeping. The admin function `update_grace_period` updates a value that is never read by any instruction, so the setting is effectively dead code and can mislead operators into thinking they are configuring an extra sweep delay that does not exist.

Recommendations:

It is recommended to remove the unused grace-period configuration entirely:

- Delete `DEFAULT_GRACE_PERIOD` and the `sweep_grace_period_seconds` field from `GlobalConfig`.
- Remove the `update_grace_period` instruction and its associated event usages.

America.fun: Resolved with `@4b469fca95 ...`

Zenith: Verified. Resolved, grace-period is used again.

[I-7] Initialization race: Anyone can become admin if protocol is not initialized immediately

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- programs/aol-staking-program/src/lib.rs

Description:

The `initialize_global_config` instruction in `programs/aol-staking-program/src/lib.rs` has no access-control beyond `payer: Signer`, and the `admin`, `operator`, `treasury`, and `penalties_wallet` public keys are freely supplied by the caller.

```
pub fn initialize_global_config(
    ctx: Context<InitializeGlobalConfig>,
    admin: Pubkey,
    operator: Pubkey,
    treasury: Pubkey,
    penalties_wallet: Pubkey,
) -> Result<()> {
    let config = &mut ctx.accounts.global_config;
    let current_time = Clock::get()?.unix_timestamp;

    config.admin = admin;
    config.pending_admin = None;
    config.operator = operator;
    config.treasury = treasury;
    config.penalties_wallet = penalties_wallet;
    config.is_paused = false;
    // ...
```

Until someone successfully calls this instruction for the first time, there is no on-chain restriction preventing an arbitrary user from:

- Invoking `initialize_global_config` on the `global_config` PDA.
- Setting themselves as `admin` and/or `operator`.
- Gaining permanent control over all admin only instructions (`pause_protocol`, `update_*`,

`admin_force_unstake`, `admin_empty_protocol_accounts`, etc.).

If deployment tooling or operational practice ever leaves the program deployed but uninitialized on mainnet even briefly, a malicious actor monitoring the deployment could front‐run the legitimate deployer and take over admin.

Recommendations:

It is recommended to bind initialization to the program's upgrade authority, i.e. add an extra signer account to `InitializeGlobalConfig` (e.g. `upgrade_authority: Signer<'info>`) and require it to match the program's upgrade authority.

America.fun: Resolved with `@91bc8c778e ...`

Zenith: Verified.

[I-8] Redundant `vault_registry` binding in reward/portfolio logic

`SEVERITY`: Informational	`IMPACT`: Informational
`STATUS`: Resolved	`LIKELIHOOD`: Low

Target

- programs/aol-staking-program/src/lib.rs#L474

Description:

In `programs/aol-staking-program/src/lib.rs`, several functions bind `let registry = &ctx.accounts.vault_registry;` but never use the variable in their logic:

- `claim_rewards`: `registry` is bound but all behavior is driven by `user` and `ctx.remaining_accounts` via `validate_vault_account`, not by any field in `VaultRegistry`.
- `get_user_portfolio`: `registry` is bound and only `registry.owner` is used (could instead read directly from `ctx.accounts.vault_registry.owner` without a separate binding).
- `get_user_claimable`: `registry` is bound, but only `registry.owner` is used to filter vaults; no other fields are referenced.

However, the unused/partially used `registry` binding is misleading:

- It suggests that registry fields like `total_user_weight` or `active_vault_indices` are involved in reward or view calculations when they are not.
- Future maintainers might incorrectly assume registry-based invariants are being enforced in `claim_rewards` when they are not.

Recommendations:

It is recommended to clean up dead or misleading bindings.

- In `claim_rewards`, remove or rename the unused variable:

```
// let registry = &ctx.accounts.vault_registry; // remove or rename to
    _registry if kept for debugging
```

- In `get_user_portfolio` / `get_user_claimable`, either:

- Use `ctx.accounts.vault_registry` directly where needed, or
- Keep the binding but ensure it is actually used for something meaningful (e.g., logging, invariants), otherwise underscore-prefix it (`let _registry = ...;`) to signal it's intentionally unused.

America.fun: Resolved with `@5b48c1d63c ...`

Zenith: Verified.

[I-9] Unused `UserClosedAllVaultsEvent` event definition

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- programs/lib_old.rs#L2765-L2772

Description:

The program defines an event:

```
#[event]
pub struct UserClosedAllVaultsEvent {
    pub user: Pubkey,
    pub vaults_closed: u8,
    pub vaults_failed: u8,
    pub total_returned: u64,
    pub timestamp: i64,
}
```

but there is no `emit!(UserClosedAllVaultsEvent { ... })` call anywhere in the codebase. All other events are actually emitted at least once. This means the event is dead code: no instruction implements a “close all vaults” operation that would populate or trigger this event.

Recommendations:

It is recommended to either remove or properly implement this event.

America.fun: Resolved with `@06e097cca8 ...`

Zenith: Verified.Resolved by removing.

[I-10] `BASIS_POINTS_DIVISOR` name is strongly misleading

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- programs/aol-staking-program/src/lib.rs#L52-L53

Description:

The constant is defined as:

```
// Basis points for multipliers (e.g., 250 = 2.5x)
pub const BASIS_POINTS_DIVISOR: u64 = 100;
```

The comment and name suggest basis points semantics, where "250" would mean 2.50% and the divisor should be 10_000. Instead, the code uses `100` and interprets "250" as 2.5× (i.e. 250%), not 2.5%. This is effectively a "percent × 1" or "multiplier in percent" scheme, not true basis points. The mismatch between the name (`BASIS_POINTS_DIVISOR`), the comment ("basis points"), and the actual value (`100`) is misleading and can easily cause incorrect assumptions when configuring multipliers or reviewing economic parameters.

Recommendations:

It is recommended to rename this constant to reflect its true semantics, e.g. `PERCENT_DIVISOR`.

America.fun: Resolved with `@4b469fca95 ...`, `@9e66a972fd ...`, `@bfb8da5557 ...` and `@f69c2f37eb ...`

Zenith: Verified. Resolved by changing the divisor from `100` to `10_000`.