



Hastra Protocol

Security Assessment

Prepared for
Figure Technology
Solutions, Inc.

Filippo Barsanti
Kevin Chow
Nicholas R. Putra
Reyn Shaw

Prepared by
Otter Audits, LLC

Prepared on
August 6th, 2026

barsa@osec.io
kchow@osec.io
nicholas@osec.io
reyn@osec.io

Table of Contents

1. Executive Summary	2
1.1. Overview	2
1.2. Key Findings	2
2. Scope	3
3. Findings	4
3.1. Severity by component	4
3.2. Findings summary	4
4. Advisories	5
<u>OS-HES-ADV-00</u> ● Incorrect NAV Application Overstates Vault Value	6
<u>OS-HES-ADV-01</u> ● Missing Staking Cooldown Around NAV Updates	7
<u>OS-HES-ADV-02</u> ● Deposit Allows Vault Token Self-Transfer	8
<u>OS-HES-ADV-03</u> ● Reusable Reward Proofs Across Epoch Versions	10
<u>OS-HES-ADV-04</u> ● Empty Administrator Lists Disable Gated Operations	12
<u>OS-HES-ADV-05</u> ● Non-Monotonic Chainlink Price Updates	14
<u>OS-HES-ADV-06</u> ● Partial Redemption Discards Outstanding Request Amount	15
<u>OS-HES-ADV-07</u> ● Missing Capped-Epoch Index Guard	16
<u>OS-HES-ADV-08</u> ● Redemption Completion Lacks Amount Binding	18
<u>OS-HES-ADV-09</u> ● Missing Vault Token Account Constraint	19
5. Suggestions	21
<u>OS-HES-SUG-00</u> ● Insufficient Fee Breaks Bulk Verification	22
<u>OS-HES-SUG-01</u> ● Lack of Administrative Recovery Path for Frozen Redemptions	23
<u>OS-HES-SUG-02</u> ● Frozen Accounts Can Fund Third-Party Deposits	24
<u>OS-HES-SUG-03</u> ● Missing Validation Logic	25
<u>OS-HES-SUG-04</u> ● Reward IDs Lack Canonical Sequence Enforcement and Replay Protection	27
<u>OS-HES-SUG-05</u> ● Possibility of Silent Truncation	28
A. Vulnerability rating scale	29
B. Procedure	30

1. Executive Summary

1.1. Overview

Figure Technology Solutions, Inc. engaged OtterSec to conduct a security assessment of the `hastra-eth-vault` and `hastra-sol-vault` programs. This assessment was conducted between June 30th and August 6th, 2026. For information on our auditing methodology, refer to [Appendix B](#).

1.2. Key Findings

We produced 16 findings throughout this audit engagement.

Among the most impactful results, we identified that the staking vault enforces no cooldown or NAV-update boundary, allowing an attacker to deposit immediately before a NAV increase and redeem afterward to capture rewards intended for existing stakers (● [OS-HES-ADV-01](#)).

We also found that `vault-mint` permits the same token account to be supplied as both the deposit source and destination when the vault authority is an externally owned signer. The resulting self-transfer leaves vault custody unchanged while receipt tokens are still minted, enabling the creation of unbacked wYLDs (● [OS-HES-ADV-02](#)).

Additionally, reward proofs are not domain-separated between legacy and V2 epochs, allowing the same allocation to be claimed through both paths when the epochs share an index and Merkle root (● [OS-HES-ADV-03](#)), and redemption completion lacks amount binding, resulting in the possibility of bypassing amount-specific compliance or operational approval (● [OS-HES-ADV-08](#)).

We further observed that a redemption request becomes irrecoverably locked if its owner is frozen, as neither normal settlement nor cancellation can complete without first thawing the account (● [OS-HES-SUG-01](#)). Also, frozen accounts can still fund deposits for unfrozen receivers because the freeze checks apply to wYLDs transfer participants rather than the transaction caller, leaving deposit funding activity outside the intended compliance restriction (● [OS-HES-SUG-02](#)).

We also advised adding additional validation within the price verification logic (● [OS-HES-SUG-03](#)), and suggested incorporating canonical sequence enforcement and replay protection for reward IDs (● [OS-HES-SUG-04](#)). Lastly, we recommended refactoring the code to mitigate silent truncation due to overflow (● [OS-HES-SUG-05](#)).

2. Scope

The `hastra-eth-vault` is an upgradeable vault system that converts USDC into wYLDs through the YieldVault and allows wYLDs to be staked through token-specific StakingVault instances for PRIME, AUTO, and SMB shares. It uses Chainlink-verified NAV data for share pricing and an asynchronous request-and-completion process for USDC redemptions.

The `hastra-sol-vault` is a two-layer Solana protocol that converts deposited assets into receipt tokens, which can then be staked through the PRIME, AUTO, and SMB staking pools for price-based yield-bearing shares.

The source code was delivered to us in Git repositories at:

1. <https://github.com/provenance-io/hastra-eth-vault> (for `hastra-eth-vault`)
2. <https://github.com/provenance-io/hastra-sol-vault> (for `hastra-sol-vault`).

`hastra-eth-vault` was audited against commit `bba117c`, and `hastra-sol-vault` was audited against commit `b094b70`. The final audit commits are `15fc053` for `hastra-eth-vault` and `4d357de` for `hastra-sol-vault`.

3. Findings

Findings were rated using the criteria in [Appendix A](#).

We split the findings into **advisories** and **suggestions**. Advisories have an immediate impact and should be remediated as soon as possible. Suggestions do not have an immediate impact but will aid in mitigating future vulnerabilities.

Items are tracked with a stable identifier: `OS-HES-ADV-*` for advisories and `OS-HES-SUG-*` for suggestions.

3.1. Severity by component

Component	● Critical	● High	● Medium	● Low	● Info	Total
hastra-evm	—	—	—	2	3	5
hastra-sol	—	—	—	8	3	11
Total	0	0	0	10	6	16

3.2. Findings summary

Severity	Count	
● Critical	0	
● High	0	
● Medium	0	
● Low	10	
● Informational	6	

4. Advisories

Here, we present a technical analysis of the 10 vulnerabilities we identified during our audit. These vulnerabilities have **immediate** security implications, and we recommend remediation as soon as possible.

ID	Severity	Status	Description
OS-HES-ADV-00	● Low (hastra-evm)	✓ Resolved	Incorrect NAV Application Overstates Vault Value
OS-HES-ADV-01	● Low (hastra-evm)	○ Ack'd	Missing Staking Cooldown Around NAV Updates
OS-HES-ADV-02	● Low (hastra-sol)	✓ Resolved	Deposit Allows Vault Token Self-Transfer
OS-HES-ADV-03	● Low (hastra-sol)	✓ Resolved	Reusable Reward Proofs Across Epoch Versions
OS-HES-ADV-04	● Low (hastra-sol)	✓ Resolved	Empty Administrator Lists Disable Gated Operations
OS-HES-ADV-05	● Low (hastra-sol)	✓ Resolved	Non-Monotonic Chainlink Price Updates
OS-HES-ADV-06	● Low (hastra-sol)	✓ Resolved	Partial Redemption Discards Outstanding Request Amount
OS-HES-ADV-07	● Low (hastra-sol)	✓ Resolved	Missing Capped-Epoch Index Guard
OS-HES-ADV-08	● Low (hastra-sol)	✓ Resolved	Redemption Completion Lacks Amount Binding
OS-HES-ADV-09	● Low (hastra-sol)	✓ Resolved	Missing Vault Token Account Constraint

Incorrect NAV Application Overstates Vault Value

ID	Severity	Status
OS-HES-ADV-00	● Low	✓ Resolved

Description

`StakingVault::getTotalValueAtNav` multiplies `totalAssets()`, which already represents the vault's managed underlying wYLDs, by the NAV and therefore applies the exchange-rate adjustment twice. Since NAV represents the underlying value per staking share, the NAV-based total value should instead be calculated as `totalSupply() * NAV / 1e18`. The current implementation can overstate the vault's reported value whenever NAV exceeds 1e18, potentially misleading accounting and monitoring integrations.

contracts/StakingVault.sol

```

1  function getTotalValueAtNav() public view returns (uint256) {
2      return Math.mulDiv(totalAssets(), getVerifiedNav(), 1e18);
3  }
```

Remediation

Calculate the NAV-based total value as `totalSupply() * NAV / 1e18`.

Patch

Resolved in [ffaca94](#).

Missing Staking Cooldown Around NAV Updates

ID	Severity	Status
OS-HES-ADV-01	● Low	○ Ack'd

Description

Because `StakingVault` prices deposits and redemptions using the current oracle NAV without enforcing a holding period or NAV-update boundary, an attacker can deposit immediately before an anticipated NAV increase and redeem immediately afterward. The lower pre-update NAV mints more shares, while the higher post-update NAV allows those shares to claim more wYLDs. With sufficient capital, this transaction-ordering strategy can capture most or all of a newly distributed reward amount intended for existing long-term stakers.

Remediation

Add an explicit timing control to `StakingVault` around NAV-sensitive staking operations.

Patch

Figure acknowledged this issue.

Deposit Allows Vault Token Self-Transfer

ID	Severity	Status
OS-HES-ADV-02	● Low	✓ Resolved

Description

The `deposit` instruction in `vault-mint` accepts separate `user_vault_token_account` and `vault_token_account` accounts, but it does not require them to be different. When `vault_authority` is an externally owned account that is also the signer, the same token account can satisfy both roles.

programs/vault-mint/src/processor.rs

```

1  pub fn deposit(ctx: Context<Deposit>, amount: u64) -> Result<> {
2      require!(!ctx.accounts.config.paused, CustomErrorCode::ProtocolPaused);
3      require!(amount > 0, CustomErrorCode::InvalidAmount);
4
5      // Validate that vault_token_account is owned by the configured vault authority
6      require!(
7          ctx.accounts.vault_token_account.owner == ctx.accounts.config.vault_authority,
8          CustomErrorCode::InvalidVaultAuthority
9      );
10
11     let cpi_accounts = Transfer {
12         from: ctx.accounts.user_vault_token_account.to_account_info(),
13         to: ctx.accounts.vault_token_account.to_account_info(),
14         authority: ctx.accounts.signer.to_account_info(),
15     };
16     token::transfer(
17         CpiContext::new(ctx.accounts.token_program.to_account_info(), cpi_accounts),
18         amount,
19     )?;
20
21     let seeds: [&[u8]] = [&[b"mint_authority", &[ctx.bumps.mint_authority]]];
22     let signer = [&seeds[..]];
23     let cpi_accounts = MintTo {
24         mint: ctx.accounts.mint.to_account_info(),
25         to: ctx.accounts.user_mint_token_account.to_account_info(),
26         authority: ctx.accounts.mint_authority.to_account_info(),
27     };
28     [...]
29     Ok(())
30 }
```

In that case, the SPL token transfer in `deposit` is a self-transfer from the account to itself. The transfer produces no balance change, but the function continues to mint the same `amount` of

receipt tokens to `user_mint_token_account`. This can allow new wYLDs to be minted without a corresponding increase in vault-token custody for that deposit path.

Remediation

Add an explicit check requiring `user_vault_token_account.key()` and `vault_token_account.key()` to differ before performing the transfer and mint.

Patch

Resolved in [PR#42](#).

Reusable Reward Proofs Across Epoch Versions

ID	Severity	Status
OS-HES-ADV-03	● Low	✓ Resolved

Description

The legacy `claim_rewards` path in `vault-mint` verifies a Merkle proof against a leaf built only from the claimant, the claimed amount, and the numeric epoch index. As shown in the snippet below, the leaf does not include a version tag or the epoch account address before the proof is checked against the legacy epoch root. After successful verification, this path mints the reward amount directly to the user's mint token account, and its one-time claim record is scoped to the legacy epoch account.

```
programs/vault-mint/src/processor.rs
```

```

1  pub fn claim_rewards(ctx: Context<ClaimRewards>, amount: u64, proof: Vec<ProofNode>)
   -> Result<()> {
2      require!(!ctx.accounts.config.paused, CustomErrorCode::ProtocolPaused);
3      require!(amount > 0, CustomErrorCode::InvalidAmount);
4      // leaf = sha256(user || amount_le || epoch_index_le)
5      let mut data = Vec::with_capacity(32 + 8 + 8);
6      data.extend_from_slice(ctx.accounts.user.key.as_ref());
7      data.extend_from_slice(&amount.to_le_bytes());
8      data.extend_from_slice(&ctx.accounts.epoch.index.to_le_bytes());
9      let mut node = hashv(&[&data]).to_bytes();
10     [...]
11 }
```

The V2 `claim_rewards_v2` path also uses the same leaf material for proof verification. The V2 accounts use a separate `epoch_v2` PDA namespace, and the claim record is again derived from the epoch account key and user key. Because the V1 and V2 epoch accounts are distinct even when they share the same numeric index, the corresponding claim-record PDAs are also distinct.

```
programs/vault-mint/src/processor.rs
```

```

1  pub fn claim_rewards_v2(ctx: Context<ClaimRewardsV2>, amount: u64, proof:
   Vec<ProofNode>) -> Result<()> {
2      [...]
3      let mut data = Vec::with_capacity(32 + 8 + 8);
4      data.extend_from_slice(ctx.accounts.user.key.as_ref());
5      data.extend_from_slice(&amount.to_le_bytes());
6      data.extend_from_slice(&ctx.accounts.epoch.index.to_le_bytes());
7      let mut node = hashv(&[&data]).to_bytes();
8
9      [...]
10 }
```

As a result, if a legacy epoch and a V2 epoch are created with the same numeric index and Merkle root, the same user, amount, and proof may satisfy both verification paths. The V1 claim record would not prevent the V2 claim record from being initialized, so the user could claim once through the legacy minting flow and once through the V2 pool-transfer flow for the same Merkle allocation.

Remediation

Domain-separate the V2 Merkle leaf from the legacy leaf. For example, include a fixed V2 prefix or version byte in the data hashed by `claim_rewards_v2` (`sha256("v2" || user || amount || index)`). This makes proofs generated for legacy epochs invalid against V2 epochs even when the numeric epoch index and allocation tuple are reused.

Patch

Resolved in [PR#42](#).

Figure removed the V2 epoch and claim path. The remaining implementation uses a single epoch and claim-record namespace, so an allocation can no longer be claimed through two versioned paths.

Empty Administrator Lists Disable Gated Operations

ID	Severity	Status
OS-HES-ADV-04	● Low	✓ Resolved

Description

In `vault-mint`, `update_freeze_administrators` allows the program update authority to replace the configured freeze administrator list, but the setter only bounds the list to at most five entries before assigning it to `config.freeze_administrators`. Because there is no lower-bound check, the update authority can submit an empty vector, leaving no key that can satisfy later `freeze_administrators.contains(...)` authorization checks.

```
programs/vault-mint/src/processor.rs
```

```

1  pub fn update_freeze_administrators(
2      ctx: Context<UpdateFreezeAdministrators>,
3      new_administrators: Vec<Pubkey>,
4  ) -> Result<()> {
5      // Validate that the signer is the program's update authority
6      validate_program_update_authority(&ctx.accounts.program_data,
7      &ctx.accounts.signer)?;
8
9
10     let config = &mut ctx.accounts.config;
11
12     require!(
13         new_administrators.len() <= 5,
14         CustomErrorCode::TooManyAdministrators
15     );
16
17     config.freeze_administrators = new_administrators;
18
19     msg!(
20         "Freeze administrators updated. New count: {}",
21         config.freeze_administrators.len()
22     );
23     Ok(())
24 }
```

`update_rewards_administrators` follows the same pattern for `config.rewards_administrators`, where it validates the update authority and enforces only the maximum count before replacing the stored list. Thus, an empty `new_administrators` value is not rejected before the rewards administrator list is overwritten.

```
programs/vault-mint/src/processor.rs
```

```

1  pub fn update_rewards_administrators(
2      ctx: Context<UpdateRewardsAdministrators>,
3      new_administrators: Vec<Pubkey>,
4  ) -> Result<()> {
5      // Validate that the signer is the program's update authority
6      validate_program_update_authority(&ctx.accounts.program_data,
7      &ctx.accounts.signer)?;
8
9      let config = &mut ctx.accounts.config;
10
11      require!(
12          new_administrators.len() <= 5,
13          CustomErrorCode::TooManyAdministrators
14      );
15
16      config.rewards_administrators = new_administrators;
17
18      msg!(
19          "Rewards administrators updated. New count: {}",
20          config.rewards_administrators.len()
21      );
22      Ok(())
23  }
```

The affected lists are used as allowlists for administrative operations in `vault-mint`. If either list is set to empty, every operation gated by that list will fail its membership check until the program update authority submits a non-empty replacement list.

Remediation

Reject empty administrator vectors in both update setters before writing to `config`, and also reject duplicate public keys.

Patch

Resolved in [PR#42](#).

Non-Monotonic Chainlink Price Updates

ID	Severity	Status
OS-HES-ADV-05	● Low	✓ Resolved

Description

`verify_price` in `vault-stake` accepts a signed Chainlink Data Streams report, checks that the caller is a rewards administrator, verifies the report through the configured Chainlink verifier accounts, confirms the report is currently within its validity window, and checks that the feed ID matches the configured feed.

After verification, the function stores the report exchange rate and sets `StakePriceConfig.price_timestamp` to the report's `observations_timestamp`. However, it does not compare that observation timestamp against the value already stored in `StakePriceConfig`, and it does not record consumed reports. As a result, a compromised rewards administrator may extract assets by submitting a valid report whose observation timestamp is equal to or older than the currently stored observation timestamp, provided the report still satisfies the validity-window and `price_max_staleness` constraint.

Remediation

Require the decoded report's `observations_timestamp` to be strictly greater than the current `stake_price_config.price_timestamp` before updating `price` and `price_timestamp`.

Patch

Resolved in [PR#43](#).

Partial Redemption Discards Outstanding Request Amount

ID	Severity	Status
OS-HES-ADV-06	● Low	✓ Resolved

Description

`complete_redeem` in `vault-mint` silently performs a partial redemption when the user's current receipt-token balance is lower than the amount recorded in the `RedemptionRequest`. By using `min(user_mint_balance, req.amount)`, it burns only the available balance and transfers the corresponding underlying assets, while Anchor still closes the entire request account. Consequently, the unfulfilled portion is discarded and cannot be completed later under the original request

```
programs/vault-mint/src/processor.rs
```

```

1  pub fn complete_redeem(ctx: Context<CompleteRedeem>) -> Result<()> {
2      // Admin gate
3      require!(
4          ctx.accounts
5              .config
6              .rewards_administrators
7              .contains(&ctx.accounts.admin.key()),
8          CustomErrorCode::InvalidRewardsAdministrator
9      );
10
11     let req = &ctx.accounts.redemption_request;
12
13     // The request redeem function will set the redeem amount to the min
14     // of the requested amount and the user's mint balance at the request.
15     // This prevents program from burning more than their balance at the time.
16     // However, we also do the same here to prevent error in the situation where
17     // the user transfers mint out of their account before this complete request
18     // executes.
19     let user_mint_balance = ctx.accounts.user_mint_token_account.amount;
20     let amount_to_redeem = std::cmp::min(user_mint_balance, req.amount);
21     require!(amount_to_redeem > 0, CustomErrorCode::InvalidAmount);
22     [...]
23 }
```

Remediation

Require the user's balance to cover the full requested amount.

Patch

Resolved in [PR#45](#).

Missing Capped-Epoch Index Guard

ID	Severity	Status
OS-HES-ADV-07	● Low	✓ Resolved

Description

Reward epoch creation in `create_rewards_epoch` does not reject indices below a capped-epoch migration boundary, allowing a rewards administrator to create an epoch at any unused index below the capped-epoch boundary. Because `claim_rewards` branches on the stored epoch index, claims for such an epoch bypass the aggregate `epoch_claimed` counter and cap enforcement. Although the declared total is checked against `max_epoch_cap`, the Merkle root is not validated against that total, so it may encode allocations whose sum is arbitrarily larger. Consequently, a malicious or compromised rewards administrator could bypass the epoch cap and mint unbounded wYLDs across distinct claimants, as each `ClaimRecord` only prevents the same user from claiming twice.

```
programs/vault-mint/src/processor.rs
```

```

1  pub fn create_rewards_epoch([...]) -> Result<()> {
2      require!(!ctx.accounts.config.paused, CustomErrorCode::ProtocolPaused);
3      require!(
4          ctx.accounts
5              .config
6              .rewards_administrators
7              .contains(&ctx.accounts.admin.key()),
8          CustomErrorCode::InvalidRewardsAdministrator
9      );
10     require!(total > 0, CustomErrorCode::InvalidAmount);
11
12     let caps = &ctx.accounts.epoch_caps_config;
13     require!(
14         total <= caps.max_epoch_cap,
15         CustomErrorCode::EpochCapAboveGlobal
16     );
17
18     let e = &mut ctx.accounts.epoch;
19     e.index = index;
20     e.merkle_root = merkle_root;
21     e.total = total;
22     e.created_ts = Clock::get()?.unix_timestamp;
23
24     ctx.accounts.epoch_claimed.claimed_total = 0;
25     [...]
26 }
```

Remediation

Reject reward epoch creation when the supplied index is less than `first_capped_epoch`.

Patch

Resolved in [PR#45](#).

Redemption Completion Lacks Amount Binding

ID	Severity	Status
OS-HES-ADV-08	● Low	✓ Resolved

Description

`complete_redeem` in `vault-mint` does not bind the administrator's approval to the redemption amount that was reviewed off-chain. Before a pending completion transaction executes, the user can cancel the original request and create a replacement with a different amount at the same user-derived `RedemptionRequest` PDA. Because the transaction references the same account address and reads its amount only at execution time, it may unknowingly complete the replacement request. Although this does not directly cause fund loss or insolvency since the corresponding wYLDs is burned and USDC is paid to the same user, it can bypass amount-specific compliance or operational approval.

Remediation

Add an `expected_amount` argument to `complete_redeem` and require it to equal `redemption_request.amount` before processing the redemption.

Patch

Resolved in [PR#45](#).

Missing Vault Token Account Constraint

ID	Severity	Status
OS-HES-ADV-09	● Low	✓ Resolved

Description

In `vault-mint`, `SweepRedeemVaultFunds` accepts the destination `vault_token_account` for sweeping tokens out of the redeem vault, but its account constraints only verify that the destination uses `config.vault` as its mint and is owned by `config.vault_authority`. As shown in the `SweepRedeemVaultFunds` account validation, the account is not compared against the configured vault token account stored in `VaultTokenAccountConfig`, so any token account satisfying the mint and owner checks may be accepted as the sweep destination.

```

programs/vault-mint/src/account_structs.rs
1  pub struct SweepRedeemVaultFunds<'info> {
2      [...]
3      #[account(
4          mut,
5          token::mint = config.vault,
6          constraint = vault_token_account.mint == config.vault @
            CustomErrorCode::InvalidVaultMint,
7          constraint = vault_token_account.owner == config.vault_authority @
            CustomErrorCode::InvalidVaultAuthority
8      )]
9      pub vault_token_account: Account<'info, TokenAccount>,
10     [...]
11 }

```

This differs from `Deposit`, which loads `vault_token_account_config` and requires the passed vault token account key to equal the configured `vault_token_account`. The missing key constraint can therefore cause sweep operations to target a vault-authority-owned account other than the configured vault token account, reducing consistency with the program's intended account configuration.

A similar issue exists in `Config`, which stores a `redeem_vault`, while the redemption flow does not validate the supplied token account against this configured address. Consequently, `complete_redeem` may accept a non-canonical token account controlled by the expected PDA and use it to fund redemptions.

Remediation

Add the `vault_token_account_config` PDA to `SweepRedeemVaultFunds`, then require `vault_token_account.key()` to equal `vault_token_account_config.vault_token_account` before allowing the sweep transfer. This keeps sweep destination validation aligned with deposit validation and ensures swept redeem-vault funds return only to the configured vault token account.

Also, enforce that `redeem_vault_token_account.key()` matches `config.redeem_vault` to bind redemptions to the designated vault.

Patch

Resolved in [PR#47](#).

5. Suggestions

Here, we present a discussion of the 6 suggestions we identified during our audit. While these findings do not present an immediate security impact, they represent anti-patterns and may result in security issues in the future.

ID	Severity	Status	Description
OS-HES-SUG-00	● Informational (hastra-evm)	✓ Resolved	Insufficient Fee Breaks Bulk Verification
OS-HES-SUG-01	● Informational (hastra-evm)	○ Ack'd	Lack of Administrative Recovery Path for Frozen Redemptions
OS-HES-SUG-02	● Informational (hastra-evm)	✓ Resolved	Frozen Accounts Can Fund Third-Party Deposits
OS-HES-SUG-03	● Informational (hastra-sol)	✓ Resolved	Missing Validation Logic
OS-HES-SUG-04	● Informational (hastra-sol)	✓ Resolved	Reward IDs Lack Canonical Sequence Enforcement and Replay Protection
OS-HES-SUG-05	● Informational (hastra-sol)	✓ Resolved	Possibility of Silent Truncation

Insufficient Fee Breaks Bulk Verification

ID	Severity	Status
OS-HES-SUG-00	● Informational	✓ Resolved

Description

`FeedVerifier::verifyBulkReports` calculates and forwards only the verification fee associated with `unverifiedReports[0]`, even though Chainlink's `FeeManager` charges the aggregate fee for every report in the batch. On fee-enabled networks, a batch containing two or more positive-fee reports is therefore underfunded, causing `VerifierProxy::verifyBulk` to revert. Since the transaction is atomic, none of the verified prices are subsequently stored, rendering bulk verification unusable for such batches. Note that the protocol currently does not pay verification fees via `FeeManager`, so bulk verification is unaffected at present.

chainlink-hub/contracts/FeedVerifier.sol

```

1  function verifyBulkReports(bytes[] calldata unverifiedReports) external whenNotPaused
   onlyRole(UPDATER_ROLE) {
2      if (unverifiedReports.length == 0) return;
3      (bytes memory parameterPayload, uint256 nativeFee) =
        _buildParameterPayload(unverifiedReports[0]);
4      bytes[] memory verifiedReports = verifierProxy.verifyBulk{value: nativeFee}(
        unverifiedReports, parameterPayload);
5      for (uint256 i = 0; i < verifiedReports.length; i++) {
6          _storeReport(verifiedReports[i]);
7      }
8  }
```

Remediation

Compute and forward the total native fee required for every report in `verifyBulkReports`, not just the first report.

Patch

Resolved in [ffaca94](#).

Lack of Administrative Recovery Path for Frozen Redemptions

ID

OS-HES-SUG-01

Severity

● Informational

Status

○ Ack'd

Description

In `YieldVault`, when an account is frozen after submitting a redemption request, `completeRedeem` explicitly rejects settlement. In contrast, `cancelRedeem` cannot return the locked shares because `_update` blocks transfers to the frozen user. Because there is no alternative resolution path, the shares remain locked in the vault unless the account is thawed.

Remediation

Introduce a narrowly authorized administrative recovery function that can cancel or settle such requests to an approved destination, with explicit accounting and event emission.

Patch

Figure acknowledged this issue.

Frozen Accounts Can Fund Third-Party Deposits

ID	Severity	Status
OS-HES-SUG-02	● Informational	✓ Resolved

Description

YieldVault enforces freezing only on the wYLDs sender and recipient processed by `_update`, rather than on the transaction caller. Consequently, a frozen account can still call `deposit`, `mint`, or `depositWithPermit`, supply `USDC`, and direct the newly minted wYLDs to an unfrozen receiver because the mint checks only that receiver. This leaves the vault's compliance freeze incomplete for deposit funding activity.

contracts/YieldVault.sol

```

1  function _update(address from, address to, uint256 amount)
2      internal
3      override(ERC20Upgradeable)
4      {
5          if (from != address(0) && frozen[from]) revert AccountIsFrozen();
6          if (to != address(0) && frozen[to]) revert AccountIsFrozen();
7          super._update(from, to, amount);
8      }

```

A similar issue exists in `StakingVault`, which applies the same `_update`-based freeze enforcement and therefore also allows a frozen caller to fund `deposit`, `mint`, or `depositWithPermit` for an unfrozen receiver.

Remediation

Reject frozen callers within the deposit path.

Patch

Resolved in [ffaca94](#) and [15fc053](#).

Missing Validation Logic

ID	Severity	Status
OS-HES-SUG-03	● Informational	✓ Resolved

Description

There are some instances within `verify_price` in `vault-stake` where proper validation logic is currently absent, as detailed below:

Firstly, `verify_price` decodes the bytes returned by `get_return_data()` but ignores the accompanying program ID, so it does not explicitly prove that the return data was produced by the configured Chainlink verifier program. Validate that the returned program ID equals `price_config.chainlink_program` before decoding and storing the report to bind the price update to the intended CPI result.

```

programs/vault-stake/src/processor.rs
1  pub fn verify_price(ctx: Context<VerifyPrice>, signed_report: Vec<u8>) -> Result<()>
2  {
3      // Authorization: signer must be a rewards administrator
4      require!(
5          ctx.accounts
6              .stake_config
7              .rewards_administrators
8              .contains(&ctx.accounts.signer.key()),
9          CustomErrorCode::InvalidRewardsAdministrator
10     );
11     [...]
12     // Decode the verified report from return data
13     let (_, return_data) =
14         get_return_data().ok_or(CustomErrorCode::ChainlinkVerifyFailed)?;
15     let report =
16         ReportDataV7::decode(&return_data).map_err(|_|
17             CustomErrorCode::ChainlinkVerifyFailed)?;
18     [...]
19 }

```

Secondly, `verify_price` only enforces `valid_from_timestamp <= observations_timestamp` and does not ensure that the observation occurred by the current time or before the report expired. A report with a future `observations_timestamp` could therefore store a future `price_timestamp`, causing downstream staleness calculations to underflow or improperly extend the price's usable lifetime. Require `observations_timestamp <= current_time` and `observations_timestamp <= expires_at` before updating `StakePriceConfig`.

```
programs/vault-stake/src/processor.rs
```

```

1  pub fn verify_price(ctx: Context<VerifyPrice>, signed_report: Vec<u8>) -> Result<()>
2  {
3      // Authorization: signer must be a rewards administrator
4      require!(
5          ctx.accounts
6              .stake_config
7              .rewards_administrators
8              .contains(&ctx.accounts.signer.key()),
9          CustomErrorCode::InvalidRewardsAdministrator
10     );
11     [...]
12     // Require consistent applicability window: valid_from is the earliest second the
13     // price
14     // applies, observations the latest (per ReportDataV7).
15     require!(
16         report.valid_from_timestamp <= report.observations_timestamp,
17         CustomErrorCode::InvalidReportTimestamps
18     );
19     [...]
20 }

```

Remediation

Implement the above-mentioned validations.

Patch

Resolved across [PR#43](#) and [PR#45](#).

Reward IDs Lack Canonical Sequence Enforcement and Replay Protection

ID	Severity	Status
OS-HES-SUG-04	● Informational	✓ Resolved

Description

In `vault-stake`, `RewardPublicationRecord` is derived from both `id` and `amount`, so publishing the same ID with a different amount produces a distinct PDA and bypasses record-based duplicate prevention. Consequently, the record only prevents replay of an identical `(id, amount)` pair rather than enforcing global uniqueness of the publication ID. If publication IDs may also advance by an unrestricted gap, an administrator could jump directly to `u32::MAX`, preventing any subsequent publication from using a larger ID.

Remediation

Add a `last_reward_publication` pda to prevent replay and require `id == last_id + 1`, or enforce a bounded `MAX_GAP` between consecutive IDs.

Patch

Resolved in [PR#45](#).

Possibility of Silent Truncation

ID	Severity	Status
OS-HES-SUG-05	● Informational	✓ Resolved

Description

The conversion helpers cast computed `u128` results to `u64` using `as`, which silently truncates values exceeding `u64::MAX` and may return an incorrect asset or share amount. Replace these casts with `try_into()`.

Remediation

Incorporate the refactor stated above.

Patch

Resolved in [PR#45](#).

Appendix A.

Vulnerability rating scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings may be found in the general findings.

Severity	Description
● Critical	Vulnerabilities that immediately result in a loss of availability, consensus divergence, or exploitable memory corruption with minimal preconditions.
● High	Vulnerabilities that may result in consensus divergence or denial of service but require specific transaction or network conditions to exploit.
● Medium	Vulnerabilities that may result in incorrect behavior or state divergence under specific configurations or edge cases.
● Low	Low probability vulnerabilities, which are still exploitable but require extenuating circumstances or undue risk.
● Informational	Best practices to mitigate future security risks. These are classified as general findings.

Resolution status is one of  Resolved,  Ack'd, or  To-do.

Appendix B.

Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the program's implementation requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of thumb, implementation vulnerabilities tend to be more checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to comprehensively understand the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that others may have missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process