our shielding · Your smart contracts, our shielding · Your smart c

shieldify

Harmonix Finance

SECURITY REVIEW

Date: 27 October 2025

CONTENTS

1. **About Shieldify Security** 3
2. **Disclaimer** 3
3. **About Harmonix Finance** 3
4. **Risk classification** 3
   - 4.1 Impact 3
   - 4.2 Likelihood 4
5. **Security Review Summary** 4
   - 5.1 Protocol Summary 4
   - 5.2 Scope 4
6. **Findings Summary** 4
7. **Findings** 5

1. About Shieldify

Positioned as the first hybrid Web3 Security company, Shieldify shakes things up with a unique subscription-based auditing model that entitles the customer to unlimited audits within its duration, as well as top-notch service quality thanks to a disruptive 6-layered security approach. The company works with very well-established researchers in the space and has secured multiple millions in TVL across protocols, also can audit codebases written in Solidity, Rust, Go, Vyper, Move and Cairo.

Learn more about us at shieldify.org.

2. Disclaimer

This security review does not guarantee bulletproof protection against a hack or exploit. Smart contracts are a novel technological feat with many known and unknown risks. The protocol, which this report is intended for, indemnifies Shieldify Security against any responsibility for any misbehavior, bugs, or exploits affecting the audited code during any part of the project's life cycle. It is also pivotal to acknowledge that modifications made to the audited code, including fixes for the issues described in this report, may introduce new problems and necessitate additional auditing.

3. About Harmonix Finance

Harmonix Finance is a DeFi platform that combines advanced hedge fund-grade strategies with native blockchain technology to optimize yield generation and liquidity efficiency. Designed for both retail and institutional investors, Harmonix offers tools to generate sustainable returns on assets like ETH, BTC, stablecoins, and DeFi positions such as staked ETH, restaked ETH, PT Pendle, and Hyperliquid tokens.

Harmonix stands apart by merging sophisticated derivative strategies, such as delta-neutral methods and out-of-the-money (OTM) options, with the transparency and accessibility of DeFi, ensuring users can earn more while mitigating risk.

4. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4.1 Impact

- **High** – results in a significant risk for the protocol's overall well-being. Affects all or most users
- **Medium** – results in a non-critical risk for the protocol affects all or only a subset of users, but is still unacceptable
- **Low** – losses will be limited but bearable – and covers vectors similar to griefing attacks that can be easily repaired

4.2 Likelihood

- **High** – almost certain to happen and highly lucrative for execution by malicious actors
- **Medium** – still relatively likely, although only conditionally possible
- **Low** – requires a unique set of circumstances and poses non-lucrative cost-of-execution to rewards ratio for the actor

5. Security Review Summary

The security review lasted 3 days, with a total of 24 hours dedicated by the Shieldify team.

Overall, the code is well-written. The audit report identified one Medium and four Low severity issues. They're related to overflow risk during vesting math with an extremely large amount, stepped vesting and others.

The Harmonix team has been highly responsive to the Shieldify research team's inquiries and promptly implemented all recommendations.

5.1 Protocol Summary

Project Name	**Harmonix Finance**
Repository	harmonix-tge
Type of Project	Yield Optimizer, Vesting, ERC20Permit
Security Review Timeline	3 days
Review Commit Hash	75f7fc4f0126e98dbb772949ee1ac48ecf2e8f23
Fixes Review Commit Hash	ff1eb668e1b770251f6bc72d551b6742d9c31045

5.2 Scope

The following smart contracts were in the scope of the security review:

File	**nSLOC**
contracts/merkle-distributor/distribution/MultiVestingDistributor.sol	131
Total	**131**

6. Findings Summary

The following number of issues have been identified, sorted by their severity:

- **Medium** issues: 1
- **Low** issues: 4

ID	Title	Severity	Status
[M-01]	Overflow Risk During Vesting Math with Extremely Large Amount (Owner/Merkle-Builder Data Issue)	Medium	Fixed
[L-01]	Stepped Vesting `Dead Period` When `periodDuration > duration`	Low	Fixed
[L-02]	The `getAvaiableAmount()` Is Misnamed and Trusts Caller-Supplied Amount	Low	Fixed
[L-03]	Inconsistent Invalid-ID Errors Across Claim Surfaces	Low	Fixed
[L-04]	Stepped Vesting `Rounding tail`: No Unlock During Final Partial Step	Low	Acknowledged

7. Findings

[M-01] Overflow Risk During Vesting Math with Extremely Large Amount (Owner/Merkle-Builder Data Issue)

Severity

Medium Risk

Description

The `getVestedAmount()` multiplies `totalGrantAmount` by BPS and by elapsed values. With Solidity 0.8 checked math, extreme `amount` values (from the Merkle leaf) can overflow and revert. That bricks `getVestedAmount()` and `claim` for that user until the root is fixed.

Location of Affected Code

File: contracts/merkle-distributor/distribution/MultiVestingDistributor.sol

```
function getVestedAmount( uint256 distributionId, uint256
    totalGrantAmount ) public view returns (uint256) {
  // code
  tgeAmount = (totalGrantAmount * config.cliffBps) / config.maxBps;
  // code
  linearVested = (totalLinearAmount * elapsedTime) / duration;
     // continuous
  linearVested = (totalLinearAmount * periodsPassed) / totalPeriods;
     // stepped
  // code
}
```

Impact

DoS for any user whose leaf encodes an oversized `amount`, they cannot claim or even query vested without revert.

Recommendation

Bound inputs or use safe scaling:

- Validate `amount` upper-bounds off-chain when building the tree and/or on-chain during creation (e.g., enforce `amount <= MAX_GRANT`).
- Or switch to 512-bit mul/div helpers (e.g., `Math.mulDiv` in OZ) where appropriate to avoid intermediate overflow.

Team Response

Fixed.

[L-01] Stepped Vesting "Dead Period" When `periodDuration > duration`

Severity

Low Risk

Description

If `periodDuration` is longer than the whole linear window (`closingTime - openingTime`), `totalPeriods` becomes `0`. During the entire open window, `linearVested` stays `0` and then jumps to the full linear amount at/after `closingTime`. This looks like a broken schedule from a user standpoint.

Location of Affected Code

File: contracts/merkle-distributor/distribution/MultiVestingDistributor.sol

```
function getVestedAmount( uint256 distributionId, uint256
    totalGrantAmount ) public view returns (uint256) {
  // code
  uint256 duration = closingTime - openingTime;
  // code
  if (config.periodDuration > 1) {
    uint256 totalPeriods = duration / config.periodDuration;
    if (totalPeriods > 0) {
      uint256 periodsPassed = elapsedTime / config.periodDuration;
      linearVested = (totalLinearAmount * periodsPassed) / totalPeriods;
    } // else: linearVested stays 0 until >= closingTime
  }
  // code
}
```

Impact

Users see no vesting at all for the whole window, then a full unlock at close. Integrations that expect stepped unlocks will display nonsense.

Recommendation

Validate `periodDuration <= (closingTime - openingTime)` in `createVesting()` / `updateVesting()`. Alternatively, auto-fallback to continuous when `duration < periodDuration` (treat as `periodDuration = 1`).

Team Response

Fixed.

[L-02] The `getAvaiableAmount()` Is Misnamed and Trusts Caller-Supplied Amount

Severity

Low Risk

Description

The public view `getAvaiableAmount()` (spelt with a typo) computes claimable using a caller-supplied `totalGrantAmount` and does not verify it against the Merkle tree. It will happily return numbers for the wrong amount, causing users confusion.

Location of Affected Code

File: contracts/merkle-distributor/distribution/MultiVestingDistributor.sol#L215

```
function getAvaiableAmount(
  uint256 distributionId,
  address account,
  uint256 totalGrantAmount
) public view returns (uint256) {
  return _hookGetClaimableAmount(distributionId, account,
     totalGrantAmount);
}
```

Impact

Off-chain callers can display incorrect "available to claim" values and then see their subsequent `claim()` revert or pay a different amount.

Recommendation

Consider applying the following changes: 1. Fix the typo (`getAvailableAmount()`). 2. Either (a) add a proof parameter and verify the leaf before returning, or (b) clearly mark as internal/for testing, and provide a separate `viewClaimable(distributionId, account, proof)` that validates against the Merkle root.

Team Response

Fixed.

[L-03] Inconsistent Invalid-ID Errors Across Claim Surfaces

Severity

Low Risk

Description

Different entry points revert with varying errors for the same invalid `distributionId`. `getDistribution()` uses `InvalidDistributionId`, while the claim path via `_hookGetMerkleRoot()` and `getVestedAmount()` bubbles `PeriodNotFound`.

Location of Affected Code

File: contracts/merkle-distributor/distribution/MultiVestingDistributor.sol

```
if (dist.merkleRoot == bytes32(0)) {
  revert InvalidDistributionId();
}

// MultiVestingDistributor._hookGetMerkleRoot
bytes32 merkleRoot = distributions[distributionId].merkleRoot;
if (merkleRoot == bytes32(0)) revert PeriodNotFound(distributionId);

// TimePeriods.getPeriod() used by getVestedAmount()
if (!periodExists(periodId)) revert PeriodNotFound(periodId);
```

Recommendation

Standardize: use `InvalidDistributionId()` for "no such distribution" everywhere and reserve `PeriodNotFound` strictly for missing time periods. Adjust `_hookGetMerkleRoot()` accordingly.

Team Response

Fixed.

[L-04] Stepped Vesting "Rounding tail": No Unlock During Final Partial Step

Severity

Low Risk

Description

When duration % periodDuration != 0', the last partial slice before `closingTime` doesn't increase `periodsPassed` (floor division). Claimable linear amount stays flat in that tail and then jumps at `closingTime`.

Location of Affected Code

File: contracts/merkle-distributor/distribution/MultiVestingDistributor.sol#L232

```
function getVestedAmount( uint256 distributionId, uint256
    totalGrantAmount ) public view returns (uint256) {
  // code
  uint256 totalPeriods = duration / config.periodDuration;
  if (totalPeriods > 0) {
    uint256 periodsPassed = elapsedTime / config.periodDuration; // floor
    linearVested = (totalLinearAmount * periodsPassed) / totalPeriods;
  }
  // code
}
```

Impact

Users see time pass with no incremental vesting near the end, then a jump at close.

Recommendation

Either (a) enforce `duration % periodDuration == 0` in create/update, or (b) handle the remainder proportionally (include a pro-rata component for the final partial step).

Team Response

Acknowledged.

shieldify

Thank you!