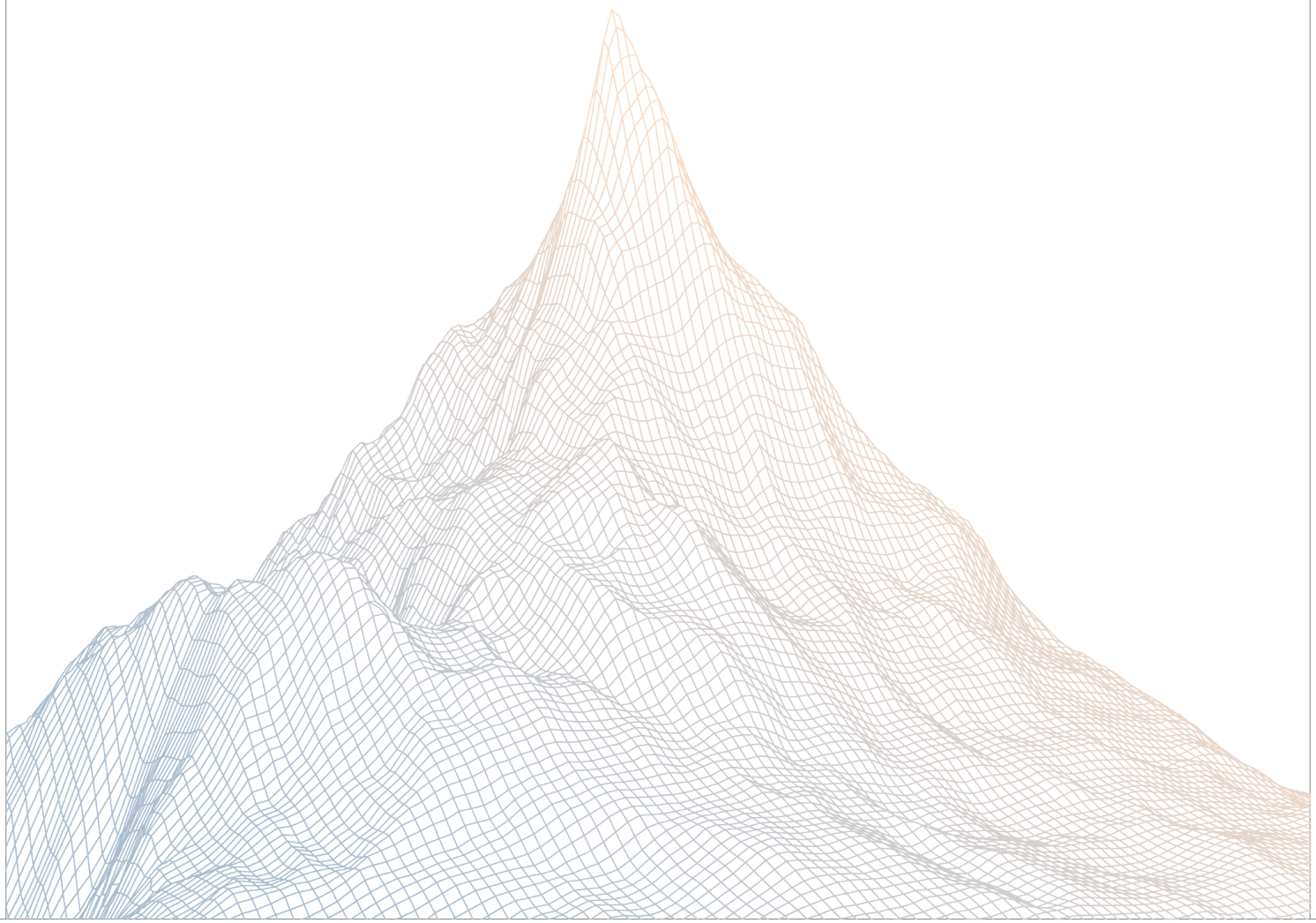
Zenith

Mitosis

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: August 27th to September 3rd, 2025

AUDITED BY: cccz
said

Contents

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 Mitosis

Mitosis introduces a protocol that transforms DeFi liquidity positions into programmable components while solving fundamental market inefficiencies. In current DeFi systems, when users provide liquidity to protocols, they encounter two significant limitations. First, their positions become static and illiquid - once assets are committed, they can't be effectively used elsewhere. Second, the most profitable opportunities remain exclusive to large investors who can negotiate private agreements, creating an uneven playing field that mirrors traditional finance systems.

2.2 Scope

The engagement involved a review of the following targets:

Target	extensible-vaults
Repository	https://github.com/mitosis-org/extensible-vaults
Commit Hash	`0a2bda5e4be799792dcaf30e662ecf6a5ce3f716`
Files	`Diff from b30eddefd1fb33ecb8fd3e02acf5b855e96d9f7d`

Target	protocol
Repository	https://github.com/mitosis-org/protocol
Commit Hash	`93345425f74fc0277c236410f22d863dcfd10049`
Files	`Diff from ec58d7f30a1903044c9030ed2c490ff2f07f811b`

2.3 Audit Timeline

August 27, 2025	Audit start
September 3, 2025	Audit end
September 21, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	1
High Risk	1
Medium Risk	5
Low Risk	4
Informational	4
Total Issues	**15**

3

Findings Summary

ID	Description	Status
C-1	Anyone can brick other users from performing claim in ReclaimQueue	Resolved
H-1	Overclaim in ReclaimQueue due to incorrect share delta when first item is not equal to reqId 0	Resolved
M-1	ReclaimQueue._sync() may fail due to insufficient balance	Resolved
M-2	Incorrect integration of reserveVLF() and previewSync()	Resolved
M-3	Interaction with entrypoints does not return excess native	Resolved
M-4	Vested tokens will be reduced when the balance is insufficient	Resolved
M-5	MitosisVaultEntrypoint._dispatchToMitosis() should call _GasRouter_dispatch() instead of _Router_dispatch()	Resolved
L-1	Too large Cap in VLFVaultCapped will block mint()	Resolved
L-2	Flawed encoding with dynamic types allows potentially passing data with incorrect values	Resolved
L-3	VLFStrategyExecutor cannot execute certain operations	Resolved
L-4	MitosisVault._setCap() may underflow	Resolved
I-1	Zero amount transfers in claim may revert	Resolved
I-2	Unsafe approve pattern in VLFStrategyExecutor	Resolved
I-3	Minor issue in ReclaimQueue._convertToAssets()	Resolved
I-4	0 amount check in AssetManager.withdraw() needs to be updated	Resolved

4

Findings

4.1 Critical Risk

A total of 1 critical risk findings were identified.

[C-1] Anyone can brick other users from performing `claim` in `ReclaimQueue`

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

Target

- ReclaimQueue.sol#L257
- ReclaimQueue.sol#L486

Description:

Anyone can call `request` for an arbitrary `receiver`. `request` accepts zero shares and does not validate the computed assets, allowing zero-asset entries to be pushed to a victim's per-recipient index.

```
function request(uint256 shares, address receiver, address vault)
    public whenNotPaused returns (uint256) {
  QueueState storage q$ = _getStorageV1().queues[vault];

  require(q$.isEnabled, IReclaimQueue__QueueNotEnabled(vault));

  IERC4626(vault).safeTransferFrom(_msgSender(), address(this), shares);

  uint256 assets = IERC4626(vault).previewRedeem(shares);

  uint48 now_ = Time.timestamp();

  uint256 reqId = q$.items.length;

  {
    uint208 assets208 = assets.toUint208();
    uint208 shares208 = shares.toUint208();

    if (reqId == 0) q$.items.push(Request(now_, assets208, shares208));
```

```
    else q$.items.push(Request(now_, assets208, q$.items[reqId
    - 1].sharesAcc + shares208));
  }

  q$.indexes[receiver].append(reqId);

  emit Requested(receiver, vault, reqId, shares, assets);

  return reqId;
}
```

`claim` reverts if the sum of assets claimed is zero. An attacker can prefill the first `MAX_CLAIM_SIZE` entries for a victim with zero-asset requests, making every claim revert and preventing progress permanently.

```
  function claim(address receiver, address vault)
    external nonReentrant whenNotPaused returns (ClaimResult memory) {
    StorageV1 storage $ = _getStorageV1();
    {
      QueueState storage q$ = $.queues[vault];
      LibQueue.UintOffsetQueue storage index = q$.indexes[receiver];

      uint32 indexSize = index.size();

      require(q$.isEnabled, IReclaimQueue__QueueNotEnabled(vault));
      require(indexSize ≠ 0, IReclaimQueue__NothingToClaim());
      require(index.offset() < indexSize, IReclaimQueue__NothingToClaim());
    }

    // run actual claim logic
    ClaimResult memory res = _claim($, receiver, vault);
>>> require(res.totalAssetsClaimed > 0, IReclaimQueue__NothingToClaim());

    emit ClaimSucceeded(receiver, vault, res);

    // send total claim amount to receiver
    IERC20Metadata(IERC4626(vault).asset()).safeTransfer(receiver,
    res.totalAssetsClaimed);

    return res;
  }
```

Recommendations:

Add input validation in `request` to ensure that the calculated shares and assets are not 0. Also, consider restricting `request` to only allow the caller as the receiver.

Mitosis: Resolved with @69b185856d..., @b28ff9a783....

Zenith: Verified.

4.2 High Risk

A total of 1 high risk findings were identified.

[H-1] Overclaim in `ReclaimQueue` due to incorrect share delta when first item is not equal to `reqId` 0

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

Target

- ReclaimQueue.sol#L400
- ReclaimQueue.sol#L447

Description:

When deciding whether to use `req.sharesAcc` instead of the delta from `reqId - 1`, the code uses `i == 0 ? req.sharesAcc : req.sharesAcc - q$.items[reqId - 1].sharesAcc`, where i is the recipient-local index, not the global `reqId`. This causes shares to be overstated for the first claimed entry when the user's first request isn't global `reqId == 0`.

```
function _execClaim(
  QueueState storage q$,
  ClaimResult memory res,
  address vault,
  address receiver,
  uint8 decimalsOffset
) internal returns (ClaimResult memory) {
  // ...
  for (uint32 i = res.reqIdFrom; i < res.reqIdTo;) {
    uint256 reqId = index.itemAt(i);
    Request memory req = q$.items[reqId];

    // if the request didn't pass the reclaim period or before the sync,
  stop the loop
    if (state.queueOffset <= reqId || state.reqTimeBoundary <
  req.timestamp) {
      res.reqIdTo = i;
```

```
        break;
      }

      if (reqId < state.cached.reqIdFrom || state.cached.reqIdTo <= reqId) {
        (state.cached, state.cachedLogPos) = _fetchSyncLogByReqId(q$,
    reqId);
      }
>>>   uint256 shares = i == 0 ? req.sharesAcc : req.sharesAcc
    - q$.items[reqId - 1].sharesAcc;
>>>   uint256 assets = Math.min(
        req.assets,
        _convertToAssets(
          shares, //
          decimalsOffset,
          state.cached.totalAssets,
          state.cached.totalSupply,
          Math.Rounding.Floor
        )
      );

     // ...
  }
```

Consider a scenario where a user requests a withdrawal and the calculated `req.assets` is 100e18. Later, when `sync` is called to fulfill the requests, the expected assets should be 80e18. But since the shares are overstated, the assets calculated by `_convertToAssets` are higher than they should be, and the user ends up receiving more assets than intended.

Recommendations:

Replace the branch with `reqId == 0` checks in both `_execClaim` and `_calcClaim`.

```
uint256 shares = i == 0 ? req.sharesAcc :
req.sharesAcc - q$.items[reqId - 1].sharesAcc;
uint256 shares = reqId == 0 ? req.sharesAcc :
req.sharesAcc - q$.items[reqId - 1].sharesAcc;
```

Mitosis: Resolved with @1891c3c8a0...

Zenith: Verified

4.3 Medium Risk

A total of 5 medium risk findings were identified.

[M-1] `ReclaimQueue._sync()` may fail due to insufficient balance

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

Target

- ReclaimQueue.sol#L555-L562

Description:

`ReclaimQueue._sync()` transfers excess shares to the ReclaimQueueCollector.

```
function _sync(StorageV1 storage $, address vault, uint256 requestCount)
    internal returns (SyncResult memory) {
  QueueState storage q$ = $.queues[vault];
  require(q$.items.length != 0, IReclaimQueue__NothingToSync());

  SyncResult memory res = _calcSync(q$, vault, requestCount);
  require(res.reqIdTo > q$.offset, IReclaimQueue__NothingToSync());

  uint256 withdrawAmount = Math.min(res.totalAssetsOnRequest,
    res.totalAssetsOnReserve);
  IERC4626(vault).withdraw(withdrawAmount, address(this), address(this));

  if (res.totalAssetsOnRequest < res.totalAssetsOnReserve) {
    uint256 assetsCollected = res.totalAssetsOnReserve
    - res.totalAssetsOnRequest;
    uint256 sharesCollected
    = IERC4626(vault).previewWithdraw(assetsCollected);

    IERC20Metadata(vault).forceApprove($.collector, sharesCollected);
    IReclaimQueueCollector($.collector).collect(vault, vault,
    sharesCollected);
    IERC20Metadata(vault).forceApprove($.collector, 0);
```

```
  }
```

The problem here is that `IERC4626.previewWithdraw()` rounds up, which can cause the calculated `sharesCollected` to be greater than the actual balance, causing the transfer to fail due to insufficient balance.

Consider a user requesting to withdraw 1000 shares. At this point, 1000 shares are worth 1000 assets. Later, `sync()` is called, and the value of 1000 shares increases to 1100 assets.

`_sync()` first withdraws 1000 assets. `withdraw()` rounds up and burns 1000/1.1 = 910 shares, leaving 90 shares. Later, when transferring the remaining shares, the 100 assets are converted to 91 shares. Due to insufficient balance, the transfer fails.

```
function _sync(StorageV1 storage $, address vault, uint256 requestCount)
    internal returns (SyncResult memory) {
  QueueState storage q$ = $.queues[vault];
  require(q$.items.length != 0, IReclaimQueue__NothingToSync());

  SyncResult memory res = _calcSync(q$, vault, requestCount);
  require(res.reqIdTo > q$.offset, IReclaimQueue__NothingToSync());

  uint256 withdrawAmount = Math.min(res.totalAssetsOnRequest,
    res.totalAssetsOnReserve);
  IERC4626(vault).withdraw(withdrawAmount, address(this), address(this));

  if (res.totalAssetsOnRequest < res.totalAssetsOnReserve) {
    uint256 assetsCollected = res.totalAssetsOnReserve
    - res.totalAssetsOnRequest;
    uint256 sharesCollected
    = IERC4626(vault).previewWithdraw(assetsCollected);

    IERC20Metadata(vault).forceApprove($.collector, sharesCollected);
    IReclaimQueueCollector($.collector).collect(vault, vault,
    sharesCollected);
    IERC20Metadata(vault).forceApprove($.collector, 0);
  }
```

Recommendations:

```
if (res.totalAssetsOnRequest < res.totalAssetsOnReserve) {
  uint256 assetsCollected = res.totalAssetsOnReserve
    - res.totalAssetsOnRequest;
uint256 sharesCollected =
IERC4626(vault).previewWithdraw(assetsCollected);
```

```
uint256 sharesCollected =
IERC4626(vault).convertToShares(assetsCollected);

  IERC20Metadata(vault).forceApprove($.collector, sharesCollected);
  IReclaimQueueCollector($.collector).collect(vault, vault,
    sharesCollected);
  IERC20Metadata(vault).forceApprove($.collector, 0);
}
```

Mitosis: Resolved with @b4dc49c789...

Zenith: Verified.

[M-2] Incorrect integration of `reserveVLF()` and `previewSync()`

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

Target

- AssetManager.sol#L217-L225
- ReclaimQueue.sol#L176-L180
- ReclaimQueue.sol#L545-L562

Description:

`reserveVLF()` requires `simulatedTotalReservedAssets ⩽ idle` to ensure there are enough funds in vlfVault.

```
function reserveVLF(address vlfVault, uint256 claimCount)
    external whenNotPaused {
  StorageV1 storage $ = _getStorageV1();

  _assertOnlyStrategist($, vlfVault);

  uint256 idle = _vlfIdle($, vlfVault);
  (, uint256 simulatedTotalReservedAssets)
    = $.reclaimQueue.previewSync(vlfVault, claimCount);
  require(simulatedTotalReservedAssets > 0,
    IAssetManager__NothingToVLFReserve(vlfVault));
  require(simulatedTotalReservedAssets <= idle,
    IAssetManager__VLFLiquidityInsufficient(vlfVault));
```

`simulatedTotalReservedAssets` will be `min(totalAssetsOnRequest,totalAssetsOnReserve)`.

```
function previewSync(address vault, uint256 requestCount)
    external view returns (uint256, uint256) {
  StorageV1 storage $ = _getStorageV1();
  SyncResult memory res = _calcSync($.queues[vault], vault, requestCount);
  return (res.totalSharesSynced, Math.min(res.totalAssetsOnRequest,
    res.totalAssetsOnReserve));
}
```

The problem here is that after modification, `sync()` no longer only withdraws `min(totalAssetsOnRequest,totalAssetsOnReserve)`, but also withdraws `totalAssetsOnReserve-totalAssetsOnRequest` to the ReclaimQueueCollector.

```
function _sync(StorageV1 storage $, address vault, uint256 requestCount)
    internal returns (SyncResult memory) {
  QueueState storage q$ = $.queues[vault];
  require(q$.items.length != 0, IReclaimQueue__NothingToSync());

  SyncResult memory res = _calcSync(q$, vault, requestCount);
  require(res.reqIdTo > q$.offset, IReclaimQueue__NothingToSync());

  uint256 withdrawAmount = Math.min(res.totalAssetsOnRequest,
    res.totalAssetsOnReserve);
  IERC4626(vault).withdraw(withdrawAmount, address(this), address(this));

  if (res.totalAssetsOnRequest < res.totalAssetsOnReserve) {
    uint256 assetsCollected = res.totalAssetsOnReserve
    - res.totalAssetsOnRequest;
    uint256 sharesCollected
    = IERC4626(vault).previewWithdraw(assetsCollected);

    IERC20Metadata(vault).forceApprove($.collector, sharesCollected);
    IReclaimQueueCollector($.collector).collect(vault, vault,
    sharesCollected);
    IERC20Metadata(vault).forceApprove($.collector, 0);
  }
```

Consider totalAssetsOnRequest == 1000, totalAssetsOnReserve == 1100, vlfVault.totalAssets() == 8000, vlfVault.allocation == 7000, so idle == 8000 - 7000 = 1000. `reserveVLF()` is called to withdraw 1000, but actually withdraws 1100. At this point, vlfVault.totalAssets() == 6900, causing `_vlfIdle()` to underflow, which invariably causes reserveVLF() and allocateVLF() to fail.

```
function _vlfIdle(StorageV1 storage $, address vlfVault)
    internal view returns (uint256) {
  uint256 total = IVLFVault(vlfVault).totalAssets();
  uint256 allocated = $.vlfStates[vlfVault].allocation;

  return total - allocated;
}
```

Recommendations:

Since :

1. When `totalAssetsOnRequest > totalAssetsOnReserve`, _sync() withdraws `totalAssetsOnReserve`.
2. When `totalAssetsOnRequest < totalAssetsOnReserve`, _sync() withdraws `totalAssetsOnRequest + (totalAssetsOnReserve - totalAssetsOnRequest) == totalAssetsOnReserve`.

So consider modifying `previewSync()` to return `totalAssetsOnReserve`.

```
function previewSync(address vault, uint256 requestCount)
    external view returns (uint256, uint256) {
  StorageV1 storage $ = _getStorageV1();
  SyncResult memory res = _calcSync($.queues[vault], vault, requestCount);
return (res.totalSharesSynced, Math.min(res.totalAssetsOnRequest, res.
    totalAssetsOnReserve));
return (res.totalSharesSynced, res.totalAssetsOnReserve );
}
```

Similarly, `previewSyncWithBudget()` should also be changed as follows

```
  function previewSyncWithBudget(address vault, uint256 budget)
    external
    view
    returns (uint256 totalSharesSynced, uint256 totalAssetsSynced,
    uint256 totalSyncedRequestsCount)
  {
    StorageV1 storage $ = _getStorageV1();
    QueueState storage q$ = $.queues[vault];

    uint32 reqIdFrom = q$.offset;
    uint32 reqIdTo = q$.items.length.toUint32();

    for (uint32 i = reqIdFrom; i < reqIdTo;) {
      Request memory req = q$.items[i];

      uint256 shares = i == 0 ? req.sharesAcc : req.sharesAcc - q$.items[i
    - 1].sharesAcc;
    uint256 assets = Math.min(req.assets, IERC4626(vault).previewRedeem(
        shares));
    uint256 assets = IERC4626(vault).previewRedeem(shares);
```

Mitosis: Resolved with @2a087ce710...

Zenith: Verified.

[M-3] Interaction with entrypoints does not return excess native

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

Target

- MitosisVault.sol#L123
- MitosisVaultDepositProxy.sol#L165
- MitosisVaultVLF.sol#L108

Description:

Several operations that interact with the entrypoint require users to provide `msg.value` for bridging.

```
function deposit(address asset, address to, uint256 amount)
    external payable whenNotPaused {
  _deposit(asset, to, amount);
  _entrypoint().deposit{ value: msg.value }(asset, to, amount,
    _msgSender());

  emit Deposited(asset, to, amount);
}
```

Inside the entrypoint, the fee is calculated, and the actual native sent equals the calculated fee.

```
  function _dispatchToMitosis(bytes memory enc, MsgType msgType,
    address refundTo) internal {
    uint96 action = uint96(msgType);

    uint256 gasLimit
    = _getHplGasRouterStorage().destinationGas[_mitosisDomain][action];
    require(gasLimit > 0, GasRouter__GasLimitNotSet(_mitosisDomain,
    action));

>>> uint256 fee = _GasRouter_quoteDispatch(_mitosisDomain, action, enc,
    address(hook()));
    _Router_dispatch(
```

```
      _mitosisDomain,
      fee,
      enc,
      StandardHookMetadata.formatMetadata(uint256(0),
      gasLimit, refundTo, bytes('')),
      address(hook())
    );
  }
```

This means the excess native is not returned and will remain stuck in the entrypoint contract. This is especially concerning for users of `MitosisVaultDepositProxy`, who expect the excess native to be refunded.

Recommendations:

Consider returning the excess native from the entrypoint.

Mitosis: Resolved with @7a237921ccd4...

Zenith: Verified. Excess fees will be handled by the configured hooks.

[M-4] Vested tokens will be reduced when the balance is insufficient

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

Target

- LinearVestingManager.sol#L114-L122

Description:

When the contract balance is insufficient, `_totalBalance()` returns the available balance.

```
function _totalBalance() internal view returns (uint256) {
  uint256 rps = _rpsHistory.latest();
  if (rps == 0) return unclaimed;

  uint256 timeDiff = block.timestamp - lastTimeClaimed;
  if (timeDiff == 0) return unclaimed;

  return Math.min(IERC20(asset).balanceOf(address(this)), unclaimed + rps
    * timeDiff);
}
```

The problem here is that in `configureVesting()` and `withdraw()`, `_totalBalance()` is directly used to modify `unclaimed`, which may cause reducing vested tokens.

```
  function configureVesting(uint256 rps)
    external onlyRole(CONFIGURE_VESTING_ROLE) {
    require(currentRps ≠ rps, RpsAlreadySet(rps));
    require(
      block.timestamp > _rpsHistory.at((_rpsHistory.length()
    - 1).toUint32())._key,
      CannotConfigureVestingInSameBlock()
    );

    // flush unapplied rewards to unclaimed
    unclaimed = _totalBalance();
...
```

```
  function withdraw(uint256 amount, address receiver) external {
    require(_msgSender() == address(vault), StdError.Unauthorized());

    // check if the amount is enough to be vested
    uint256 vested = _totalBalance();
    require(amount <= vested, InsufficientBalance(vested, amount));

    unclaimed = vested - amount;
```

Assuming the user has 600 tokens vested, but the contract balance is 500. When the user calls `withdraw()` to withdraw 500 tokens, `unclaimed` is set to 0 and `lastTimeClaimed` is set to block.timestamp. This results in the user losing the vested 100 tokens.

Recommendations:

It is recommended to no longer use the contract balance to limit the return value of `_totalBalance()`, and modify `totalBalance()` and `withdraw()` as follows.

```
diff --git a/src/managers/LinearVestingManager.sol
    b/src/managers/LinearVestingManager.sol
index 0b571bb..040d7b1 100644
-- a/src/managers/LinearVestingManager.sol
++ b/src/managers/LinearVestingManager.sol
@@ -57,7 +57,7 @@ contract LinearVestingManager is IManager,
    AccessControlEnumerable {
   }

   function totalBalance() external view returns (uint256) {
    return _totalBalance();

     return Math.min(IERC20(asset).balanceOf(address(this)), _totalBalance());
   }

   function rpsAt(uint256 timestamp) external view returns (uint256) {
@@ -82,7 +82,8 @@ contract LinearVestingManager is IManager,
    AccessControlEnumerable {

     // check if the amount is enough to be vested
     uint256 vested = _totalBalance();
    require(amount <= vested, InsufficientBalance(vested, amount));
    uint256 bal = IERC20(asset).balanceOf(address(this));
    require(amount <= bal, InsufficientBalance(bal, amount));
```

```
      unclaimed = vested - amount;
      paid += amount;
@@ -118,6 +119,6 @@ contract LinearVestingManager is IManager,
     AccessControlEnumerable {
      uint256 timeDiff = block.timestamp - lastTimeClaimed;
      if (timeDiff == 0) return unclaimed;


-     return Math.min(IERC20(asset).balanceOf(address(this)), unclaimed + rps *
          timeDiff);
+     return unclaimed + rps * timeDiff;
    }
  }
```

Mitosis: Resolved with PR-40

Zenith: Verified.

[M-5] `MitosisVaultEntrypoint._dispatchToMitosis()` should call `_GasRouter_dispatch()` instead of `_Router_dispatch()`

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

Target

- MitosisVaultEntrypoint.sol#L172-L186

Description:

`MitosisVaultEntrypoint._dispatchToMitosis()` calls `_GasRouter_quoteDispatch()` to quote gas, but calls `_Router_dispatch()` to dispatch messages.

```
function _dispatchToMitosis(bytes memory enc, MsgType msgType,
    address refundTo) internal {
  uint96 action = uint96(msgType);

  uint256 gasLimit
    = _getHplGasRouterStorage().destinationGas[_mitosisDomain][action];
  require(gasLimit > 0, GasRouter__GasLimitNotSet(_mitosisDomain, action));

  uint256 fee = _GasRouter_quoteDispatch(_mitosisDomain, action, enc,
    address(hook()));
  _Router_dispatch(
    _mitosisDomain,
    fee,
    enc,
    StandardHookMetadata.formatMetadata(uint256(0), gasLimit, refundTo,
    bytes('')),
    address(hook())
  );
}
```

However, their `_hookMetadata` is different, which may cause incorrect quoted Gas or affect cross-chain execution due to different `_hookMetadata`.

```
  function _GasRouter_dispatch(
    uint32 _destination,
```

```
    uint96 _action,
    uint256 _value,
    bytes memory _messageBody,
    address _hook
  ) internal returns (bytes32) {
    return _Router_dispatch(_destination, _value, _messageBody,
    _GasRouter_hookMetadata(_destination, _action), _hook);
  }
...
  function _GasRouter_hookMetadata(uint32 _destination, uint96 _action)
    internal view returns (bytes memory) {
    uint256 gasLimit
    = _getHplGasRouterStorage().destinationGas[_destination][_action];
    // IGP does not overrides gas limit even if it is set to zero.
    // So we need to check if the gas limit is properly set.
    require(gasLimit > 0, GasRouter__GasLimitNotSet(_destination, _action));
    return StandardHookMetadata.overrideGasLimit(gasLimit);
  }
```

Recommendations:

```
function _dispatchToMitosis(bytes memory enc, MsgType msgType,
    address refundTo) internal {
  uint96 action = uint96(msgType);

  uint256 gasLimit
    = _getHplGasRouterStorage().destinationGas[_mitosisDomain][action];
  require(gasLimit > 0, GasRouter__GasLimitNotSet(_mitosisDomain, action));

  uint256 fee = _GasRouter_quoteDispatch(_mitosisDomain, action, enc,
    address(hook()));
_Router_dispatch(
_GasRouter_dispatch(
    _mitosisDomain,
   fee,
   action,
   fee,
    enc,

    StandardHookMetadata.formatMetadata(uint256(0), gasLimit, refundTo, bytes(
      '')),
    address(hook())
  );
```

```
}
```

Mitosis: Resolved with @c709d7b0ab...

Zenith: Verified.

4.4 Low Risk

A total of 4 low risk findings were identified.

[L-1] Too large Cap in VLFVaultCapped will block `mint()`

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

Target

- VLFVaultCapped.sol#L97-L99

Description:

VLFVaultCapped.maxMint() uses Cap to calculate the maximum mintable amount.

```
  function maxMint(address account)
    public view override returns (uint256 maxShares) {
    return convertToShares(maxDeposit(account));
  }
...
  function maxDeposit(address)
    public view virtual override returns (uint256 maxAssets) {
    return _maxDepositForAllCaps(_getVLFVaultCappedStorage());
  }
```

If the protocol chooses type(uint256).max as Cap to indicate unlimited minting, this will cause overflow in `maxMint()`(even though `fullMulDiv()` avoids overflow in x * y, it cannot avoid overflow in x * y / z, i.e., `fullMulDiv(2**256-1,1e24,1e18)` will overflow).

```
function convertToShares(uint256 assets)
    public view virtual returns (uint256 shares) {
    if (!_useVirtualShares()) {
        uint256 supply = totalSupply();
        return _eitherIsZero(assets, supply)
            ? _initialConvertToShares(assets)
            : FixedPointMathLib.fullMulDiv(assets, supply, totalAssets());
    }
```

```
    uint256 o = _decimalsOffset();
    if (o == uint256(0)) {
        return FixedPointMathLib.fullMulDiv(assets, totalSupply() + 1,
    _inc(totalAssets()));
    }
    return FixedPointMathLib.fullMulDiv(assets, totalSupply() + 10 ** o,
    _inc(totalAssets()));
}
```

This will block `mint()`.

```
function mint(uint256 shares, address to)
    public virtual returns (uint256 assets) {
    if (shares > maxMint(to)) _revert(0x6a695959); // `MintMoreThanMax()`.
    assets = previewMint(shares);
    _deposit(msg.sender, to, assets, shares);
}
```

Recommendations:

It is recommended to set an upper limit for Cap.

Mitosis: Resolved with @f019799f52...

Zenith: Verified.

[L-2] Flawed encoding with dynamic types allows potentially passing data with incorrect values

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

Target

- MerkleRewardDistributor.sol#L341-L349

Description:

The message hash is currently constructed as `keccak256(bytes.concat(keccak256(abi.encodePacked(receiver, stage, vault, rewards, amounts))))`

```
function _leaf(address receiver, uint256 stage, address vault, address[]
    calldata rewards, uint256[] calldata amounts)
  internal
  pure
  returns (bytes32 leaf)
{
  // double-hashing to prevent second preimage attacks:
  // https://flawed.net.nz/2018/02/21/attacking-merkle-trees-with-a-second-
    preimage-attack/
  return keccak256(bytes.concat(keccak256(abi.encodePacked(receiver, stage,
    vault, rewards, amounts))));
}
```

since `rewards` and `amounts` are of dynamic type, the same final encoded bytes can be attributed to multiple combinations and potential collisions.

Recommendations:

Use hash version of `rewards` and `amounts`.

```
function _leaf(address receiver, uint256 stage, address vault, address[]
    calldata rewards, uint256[] calldata amounts)
  internal
```

```
  pure
  returns (bytes32 leaf)
{
  // double-hashing to prevent second preimage attacks:
  // https://flawed.net.nz/2018/02/21/attacking-merkle-trees-with-a-second-
    preimage-attack/
  bytes32 rewardsHash = keccak256(abi.encode(rewards));
  bytes32 amountsHash = keccak256(abi.encode(amounts));
  return keccak256(bytes.concat
  (keccak256(abi.encodePacked(receiver, stage, vault, rewards, amounts))));
  return keccak256(bytes.concat(keccak256(abi.encodePacked
  (receiver, stage, vault, rewardsHash, amountsHash))));
}
```

Mitosis: Resolved with @cf3b2ab248...

Zenith: Verified.

[L-3] `VLFStrategyExecutor` cannot execute certain operations

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

Target

- VLFStrategyExecutor.sol#L174-L200

Description:

The `execute` functions in `VLFStrategyExecutor` allow performing arbitrary operations on a `target` and providing `value` as native to the target. However, these functions do not have the `payable` modifier, and `VLFStrategyExecutor` is not designed to hold native tokens. As a result, operations that require sending native to the target are not possible.

```
function execute(address target, bytes calldata data, uint256 value)
  external
  nonReentrant
  returns (bytes memory result)
{
  StorageV1 memory $ = _getStorageV1();
  _assertOnlyExecutor($);

  result = target.functionCallWithValue(data, value);
}

function execute(address[] calldata targets, bytes[] calldata data,
    uint256[] calldata values)
  external
  nonReentrant
  returns (bytes[] memory results)
{
  require(targets.length == data.length && data.length == values.length,
    StdError.InvalidParameter('executeData'));

  StorageV1 memory $ = _getStorageV1();
  _assertOnlyExecutor($);

  uint256 targetsLength = targets.length;
  results = new bytes[](targetsLength);
```

```
  for (uint256 i; i < targetsLength; ++i) {
    results[i] = targets[i].functionCallWithValue(data[i], values[i]);
  }
}
```

Recommendations:

Add the `payable` modifier to the `execute` functions.

Mitosis: Resolved with @8c878d2c92...

Zenith: Verified.

[L-4] `MitosisVault._setCap()` may underflow

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

Target

- MitosisVault.sol#L206-L216

Description:

`MitosisVault._setCap()` may underflow when calculating `prevSpent` because `availableCap` may exceed `maxCap`.

```
function _setCap(StorageV1 storage $, address asset, uint256 newCap)
    internal {
  AssetInfo storage assetInfo = $.assets[asset];

  uint256 prevCap = assetInfo.maxCap;
  uint256 prevSpent = prevCap - assetInfo.availableCap;

  assetInfo.maxCap = newCap;
  assetInfo.availableCap = newCap - Math.min(prevSpent, newCap);

  emit CapSet(_msgSender(), asset, prevCap, newCap);
}
```

Consider the following scenario:

1. LIQUIDITY_MANAGER_ROLE set maxCap 100, availableCap is 100.
2. User deposit 60, availableCap is 40.
3. LIQUIDITY_MANAGER_ROLE set maxCap 50, availableCap = 50 - min(60,50) = 0.
4. User withdraw 60, availableCap = 60.
5. LIQUIDITY_MANAGER_ROLE call `setCap()`, maxCap - availableCap = 50 - 60 will underflow.

```
function withdraw(address asset, address to, uint256 amount)
    external whenNotPaused {
  StorageV1 storage $ = _getStorageV1();

  _assertOnlyEntrypoint($);
  _assertAssetInitialized(asset);

  $.assets[asset].availableCap += amount;

  IERC20(asset).safeTransfer(to, amount);

  emit Withdrawn(asset, to, amount);
}
```

Recommendations:

```
function _setCap(StorageV1 storage $, address asset, uint256 newCap)
    internal {
  AssetInfo storage assetInfo = $.assets[asset];

  uint256 prevCap = assetInfo.maxCap;
- uint256 prevSpent = prevCap - assetInfo.availableCap;
+ uint256 prevSpent = prevCap - Math.min(prevCap,assetInfo.availableCap);

  assetInfo.maxCap = newCap;
  assetInfo.availableCap = newCap - Math.min(prevSpent, newCap);

  emit CapSet(_msgSender(), asset, prevCap, newCap);
}
```

Mitosis: Resolved with @9ce97be2faf...

Zenith: Verified.

4.5 Informational

A total of 4 informational findings were identified.

[I-1] Zero amount transfers in `claim` may revert

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

Target

- MerkleRewardDistributor.sol#L331

Description:

Inside `MerkleRewardDistributor`, the claim flow unconditionally calls transfer for each reward, including zero-amount entries. Some non-standard ERC20s revert on zero-value transfers, which would revert the entire claim (including other non-zero rewards).

```
function _claim(
  address receiver,
  uint256 stage,
  address vault,
  address[] calldata rewards,
  uint256[] calldata amounts,
  bytes32[] calldata proof
) internal {
  StorageV1 storage $ = _getStorageV1();
  Stage storage s = _stage($, stage);
  require(!s.claimed[receiver][vault],
  IMerkleRewardDistributor__AlreadyClaimed());

  uint256 rewardsLen = rewards.length;
  bytes32 leaf = _leaf(receiver, stage, vault, rewards, amounts);
  require(proof.verify(s.root, leaf),
  IMerkleRewardDistributor__InvalidProof());
  require(rewardsLen == amounts.length,
  StdError.InvalidParameter('amounts.length'));
  s.claimed[receiver][vault] = true;
  for (uint256 i = 0; i < rewardsLen; i++) {
```

```
      $.reservedRewardAmounts[rewards[i]] -= amounts[i];
    }

    for (uint256 i = 0; i < rewardsLen; i++) {
>>>   IERC20(rewards[i]).safeTransfer(receiver, amounts[i]);
    }

    emit Claimed(receiver, stage, vault, rewards, amounts);
  }
```

Recommendations:

In `_claim`, skip transfers when amount is 0.

Mitosis: Resolved with @cc4e4b8dcf...

Zenith: Verified.

[I-2] Unsafe approve pattern in `VLFStrategyExecutor`

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

Target

- VLFStrategyExecutor.sol#L138
- VLFStrategyExecutor.sol#L168

Description:

`returnLiquidity` and `settleExtraRewards` use unsafe `approve`, which can cause issues. Tokens that require resetting the allowance to 0 before setting a new amount will revert.

```
function returnLiquidity(uint256 amount) external {
  require(amount > 0, StdError.ZeroAmount());

  StorageV1 storage $ = _getStorageV1();

  _assertOnlyStrategist($);

  $.asset.approve(address($.vault), amount);
  $.vault.returnVLF($.hubVLFVault, amount);
  $.storedTotalBalance -= amount;
}
```

Recommendations:

Use `SafeERC20`'s forceApprove.

Mitosis: Resolved with @43f316ef62...

Zenith: Verified.

[I-3] Minor issue in `ReclaimQueue._convertToAssets()`

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

Target

- ReclaimQueue.sol#L311-L319

Description:

`ReclaimQueue._convertToAssets()` is used to calculate the Assets claimed by the user.

```
      uint256 assets = Math.min(
        req.assets,
        _convertToAssets(
          shares, //
          decimalsOffset,
          state.cached.totalAssets,
          state.cached.totalSupply,
          Math.Rounding.Floor
        )
      );
...
  function _convertToAssets(
    uint256 shares,
    uint8 decimalsOffset,
    uint256 totalAssets,
    uint256 totalSupply,
    Math.Rounding rounding
  ) private pure returns (uint256) {
    return shares.mulDiv(totalAssets, totalSupply + 10 ** decimalsOffset,
    rounding);
  }
```

It is slightly different from `ERC4626.convertToAssets()` in that it requires adding 1 to totalAssets.

```
function convertToAssets(uint256 shares)
    public view virtual returns (uint256 assets) {
```

```
    if (!_useVirtualShares()) {
        uint256 supply = totalSupply();
        return supply == uint256(0)
            ? _initialConvertToAssets(shares)
            : FixedPointMathLib.fullMulDiv(shares, totalAssets(), supply);
    }
    uint256 o = _decimalsOffset();
    if (o == uint256(0)) {
        return FixedPointMathLib.fullMulDiv(shares, totalAssets() + 1,
    _inc(totalSupply()));
    }
    return FixedPointMathLib.fullMulDiv(shares, totalAssets() + 1,
    totalSupply() + 10 ** o);
}
```

Recommendations:

```
function _convertToAssets(
  uint256 shares,
  uint8 decimalsOffset,
  uint256 totalAssets,
  uint256 totalSupply,
  Math.Rounding rounding
) private pure returns (uint256) {
return shares.mulDiv(totalAssets, totalSupply + 10 ** decimalsOffset,
    rounding);
return shares.mulDiv(totalAssets + 1, totalSupply + 10 ** decimalsOffset,
    rounding);
}
```

Mitosis: Resolved with @1569439e6b...

Zenith: Verified.

[I-4] 0 amount check in `AssetManager.withdraw()` needs to be updated

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

Target

- AssetManager.sol#L405-L414

Description:

`_scaleToBranchDecimals()` may cause a non-zero `amount` to be returned as 0, so it should check that the new `amount` is not 0.

```
function withdraw(uint256 chainId, address hubAsset, address to,
    uint256 amount) external payable whenNotPaused {
  StorageV1 storage $ = _getStorageV1();

  require(to ≠ address(0), StdError.ZeroAddress('to'));
  require(amount ≠ 0, StdError.ZeroAmount());

  address branchAsset = _hubAssetState($, hubAsset, chainId).branchAsset;
  _assertBranchAssetPairExist($, chainId, branchAsset);

  uint256 amountBranchUnit;
  (amountBranchUnit, amount) = _scaleToBranchDecimals(
    amount, _hubAssetState($, hubAsset, chainId).branchAssetDecimals,
    IHubAsset(hubAsset).decimals()
  );

  _assertBranchAvailableLiquiditySufficient($, hubAsset, chainId, amount);
  _assertBranchLiquidityThresholdSatisfied($, hubAsset, chainId, amount);

  _burn($, chainId, hubAsset, _msgSender(), amount);
  $.entrypoint.withdraw{ value: msg.value }(chainId, branchAsset, to,
    amountBranchUnit);

  emit Withdrawn(chainId, hubAsset, to, amount, amountBranchUnit);
}
```

Recommendations:

```
function withdraw(uint256 chainId, address hubAsset, address to,
    uint256 amount) external payable whenNotPaused {
  StorageV1 storage $ = _getStorageV1();

  require(to ≠ address(0), StdError.ZeroAddress('to'));
require(amount ≠ 0, StdError.ZeroAmount());

  address branchAsset = _hubAssetState($, hubAsset, chainId).branchAsset;
  _assertBranchAssetPairExist($, chainId, branchAsset);

  uint256 amountBranchUnit;
  (amountBranchUnit, amount) = _scaleToBranchDecimals(
    amount, _hubAssetState($, hubAsset, chainId).branchAssetDecimals,
    IHubAsset(hubAsset).decimals()
  );
require(amount ≠ 0, StdError.ZeroAmount());
```

Mitosis: Resolved with @8a521b22a9...

Zenith: Verified.