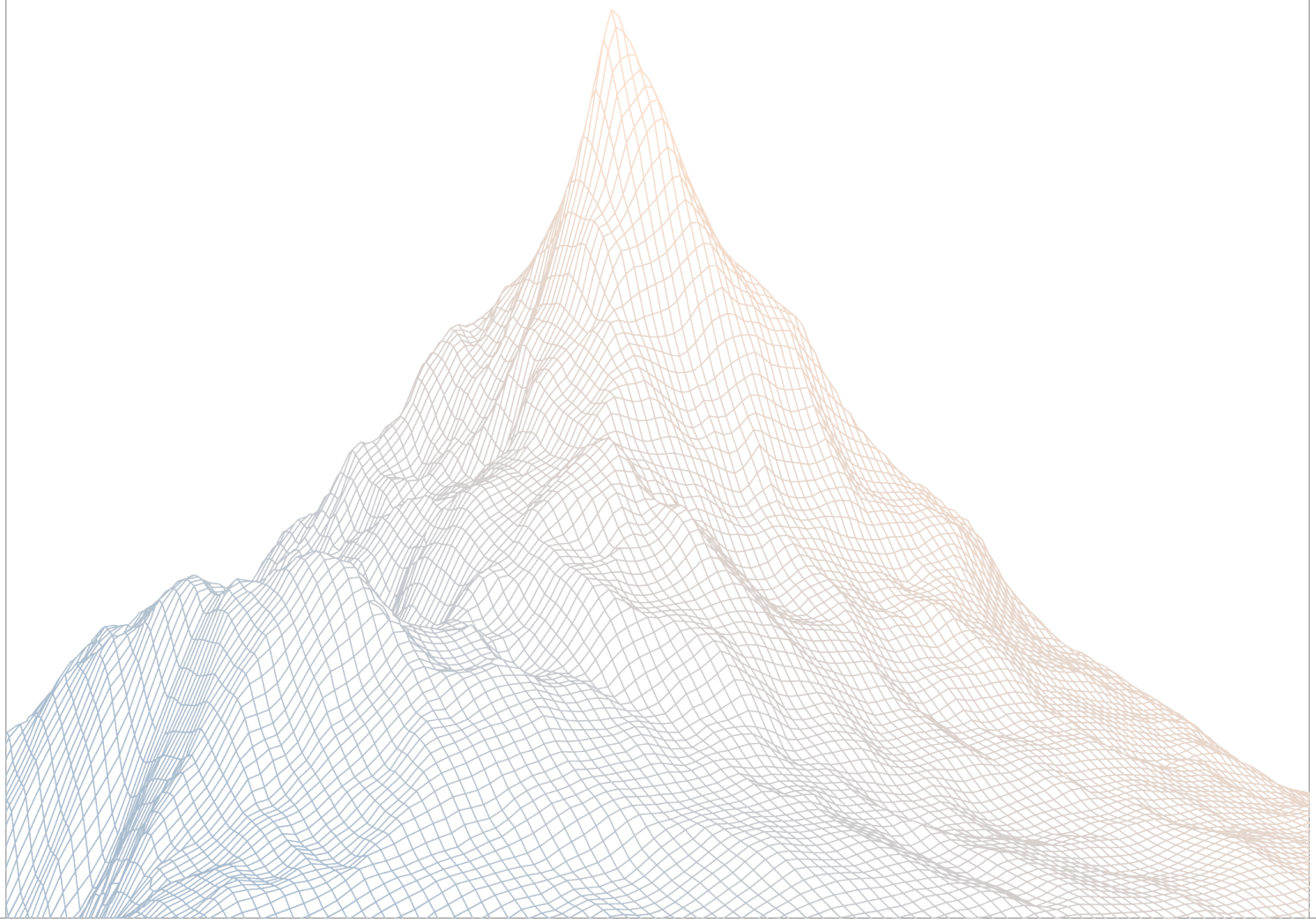
Zenith

Moonbirds

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: January 24th to January 28th, 2026

AUDITED BY: Mario Poneder
oakcobalt

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 Moonbirds

Moonbirds is more than just a collection of digital owls; we are a global community of collectors, builders, and innovators. As the creators of the Vibes TCG and the $BIRB ecosystem, we are obsessed with blending physical toys with digital experiences. Our events are designed to connect the "flock" in real life, offering a space to network, celebrate, and get an exclusive look at the future of our brand. Whether you're a long-time holder or new to the nest, we welcome you to experience the culture we are building.

2.2 Scope

The engagement involved a review of the following targets:

Target	birb-contract
Repository	https://github.com/Moonbirds-OCG/birb-contract.git
Commit Hash	`19f6c893b5845b943f3533e3225162bd77b2a809`
Files	`src/MoonbirdsEscrowAdapter.sol`

2.3 Audit Timeline

January 24, 2026	Audit start
January 28, 2026	Audit end
January 29, 2026	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	0
Medium Risk	3
Low Risk	4
Informational	5
Total Issues	**12**

3

Findings Summary

ID	Description	Status
M-1	Missing original owner/solanaAddress Validation in lzCompose	Resolved
M-2	lzCompose may cause user excess token loss due to missing refund	Resolved
M-3	Missing Source Endpoint ID Validation	Resolved
L-1	Users can lose funds when nesting/redemption occurs before rate lock	Acknowledged
L-2	nest() DoS due to OFT fee charges or rounding logic	Acknowledged
L-3	nest forwards user-supplied LayerZero extraOptions without enforcement, risking underfunded destination execution	Acknowledged
L-4	Deployer implicitly owns onlyOwner privileges while _owner only sets admin and endpoint delegate	Resolved
I-1	For-loops can be optimized to save gas in releaseWithTokensBatch	Resolved
I-2	Unused import	Resolved
I-3	depositsEnabled defaults to true and is not tied to rate finalization	Resolved
I-4	nest uses msg.sender as the LayerZero refund address	Acknowledged
I-5	lzCompose failures can leave NFTs escrowed and funds stranded	Acknowledged

4

Findings

4.1 Medium Risk

A total of 3 medium risk findings were identified.

[M-1] Missing original owner/solanaAddress Validation in lzCompose

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

Target

- MoonbirdsEscrowAdapter.sol#L239-L243

Description:

The `lzCompose()` function does not validate that the source chain initiator (the address on Solana that initiated the cross-chain redemption) is the original owner of the NFT.

While the NFT is always released to `record.owner` (the original depositor on Ethereum), anyone on Solana can initiate a redemption for any NFT by paying the required tokens. This allows unauthorized parties to force early redemptions, disrupting user nesting strategies and potentially affecting off-chain economic calculations that depend on nesting duration (`nestedAt` timestamp).

```
function lzCompose(
    address _from,
    bytes32 _guid,
    bytes calldata _message,
    address _executor,
    bytes calldata _extraData
) external payable override whenNotPaused nonReentrant {
    // Only endpoint can call this
    if (msg.sender ≠ address(endpoint)) revert OnlyEndpoint();

    // Verify the compose came from our BirbOFT
    if (_from ≠ address(birbOft)) revert InvalidComposeFrom();

    // Decode the OFT compose message header
    uint256 amountReceived = OFTComposeMsgCodec.amountLD(_message);
```

```
    bytes memory composeMsg = OFTComposeMsgCodec.composeMsg(_message);

    // Decode our custom compose message: (collection, tokenId)
    (uint8 collectionId, uint256 tokenId) = abi.decode(
        composeMsg,
        (uint8, uint256)
    );

    // @audit:  No validation of composeFrom (source chain initiator)  is the
    original owner of the NFT
}
```

In contrast, the alternative `releaseWithTokens` and `releaseWithTokensBatch` only allow the original owner of the NFT to unnest.

Impacts: Unauthorized early redemption. Other users can force redemption/unnesting of NFTs they don't own, reducing others' NFT's nesting duration, which might cause the target users' economic impacts if nesting duration is tied to reward mechanisms.

Recommendations:

1. Consider storing solana address in EscrowRecord.
2. In `lzCompose`, extract the source chain initiator through `OFTComposeMsgCodec.composeFrom(_message)` and validate that the initiator matches the recorded solana address. For example,

```
function lzCompose(...)
    external payable override whenNotPaused nonReentrant {
...
    bytes32 composeFrom = OFTComposeMsgCodec.composeFrom(_message); //
    Extract source chain initiator
...
    if (composeFrom ≠ record.solanaAddress) {
        //revert
    }
```

Moonbirds: Resolved with @a8e491f ...

Zenith: Verified.

[M-2] lzCompose may cause user excess token loss due to missing refund

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

Target

- MoonbirdsEscrowAdapter.sol#L253

Description:

The `lzCompose` function does not refund excess tokens when `amountReceived` exceeds `requiredAmount`. This can occur when:

1. OFT fees are reduced on the source chain (Solana) between when users calculate the required amount and when their transaction executes.
2. Rounding discrepancies during decimal conversions between chains.

Excess tokens remain permanently locked in the contract with no recovery mechanism.

```
    function lzCompose(
...
    ) external payable override whenNotPaused nonReentrant {
...
uint256 requiredAmount = _getTokenAmount(collection) + redemptionFee;
if (amountReceived < requiredAmount)
    revert InsufficientTokensReceived();
```

Suppose this scenario below: T0: User calculates redemption amount

- Required: 10,000 BIRB (rate) + 100 BIRB (redemption fee) = 10,100 total
  - Current OFT fee on Solana: 1% (100 BPS)
  - User calculates and sends tx: send 10,201 BIRB

T1: OFT owner reduces fee to 0.5% T2: User tx settles on Solana: Message contains: 10,150 BIRB T3: On Ethereum, lzReceive credits 10150 BIRB T4: lzCompose doesn't refund the excess 50 BIRB.

Recommendations:

Consider adding excess token check and refund in lzCompose. For example,

```
if (amountReceived > requiredAmount) {
    uint256 excess = amountReceived - requiredAmount;
    address tokenAddr = birbOft.token();
    IERC20(tokenAddr).safeTransfer(record.owner, excess);
    emit ExcessRefunded(record.owner, collection, tokenId, excess);
}
```

Moonbirds: Resolved with @a8e491f ...

Zenith: Verified.

[M-3] Missing Source Endpoint ID Validation

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

Target

- MoonbirdsEscrowAdapter.sol#L231-L234

Description:

The `lzCompose()` method in MoonbirdsEscrowAdapter does not validate that the original cross-chain message came from the expected source endpoint(`srcEid`). `srcEid` is encoded in `_message` but not checked.

```
function lzCompose(
    address _from,
    bytes32 _guid,
    bytes calldata _message,
    address _executor,
    bytes calldata _extraData
) external payable override whenNotPaused nonReentrant {
    // Only endpoint can call this
    if (msg.sender != address(endpoint)) revert OnlyEndpoint();

    // Verify the compose came from our BirbOFT
    if (_from != address(birbOft)) revert InvalidComposeFrom();

    // Decode the OFT compose message header
    uint256 amountReceived = OFTComposeMsgCodec.amountLD(_message);
    bytes memory composeMsg = OFTComposeMsgCodec.composeMsg(_message);
    //@audit The srcEid is encoded in _message but not checked
```

Impacts: Case1: If `birbOft` is configured to accept messages from multiple chains, this allows other chains to call `lzCompose` on the escrow adapter. Case2: if `solanaEid` has been updated to a new endpoint, this still accepts messages sent from an outdated endpoint. For example,

1. birbOft on Ethereum allows enabled to peer for `lzReceive()`
2. Exectuor calls endpoint.lzCompose() -> adapter.lzCompose()

3. Escrow adapter accepts it because it satisfies `msg.sender` and `_from` check. But the message is not coming from the expected Solana chain or expected endpoint.

Recommendations:

Consider adding `srcEid` validation to `lzCompose()`:

```
function lzCompose(
    address _from,
    bytes32 _guid,
    bytes calldata _message,
    address _executor,
    bytes calldata _extraData
) external payable override whenNotPaused nonReentrant {
    if (msg.sender ≠ address(endpoint)) revert OnlyEndpoint();
    if (_from ≠ address(birbOft)) revert InvalidComposeFrom();
    uint256 amountReceived = OFTComposeMsgCodec.amountLD(_message);
    uint32 srcEid = OFTComposeMsgCodec.srcEid(_message);  // Extract srcEid
    bytes memory composeMsg = OFTComposeMsgCodec.composeMsg(_message);
    // Validate that compose messages only come from Solana
    if (srcEid ≠ solanaEid) revert InvalidComposeSender();
...
}
```

Moonbirds: Resolved with @a8e491f ...

Zenith: Verified.

4.2 Low Risk

A total of 4 low risk findings were identified.

[L-1] Users can lose funds when nesting/redemption occurs before rate lock

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

Target

- MoonbirdsEscrowAdapter.sol#L180

Description:

The contract doesn't enforce that `nest()` is only allowed after token rates are locked. And deposit is enabled atomically at contract deployment, allowing nesting immediately while token rates are allowed to be tuned further per documentation.

This creates risks of inconsistent token price for the same collection and user fund loss.

1. In `nest()`, `tokenRates.locked` is not checked.

```
function nest(uint256 tokenId, Collection collection, bytes32 solanaAddress,
    bytes calldata extraOptions)
    external
    payable
    whenNotPaused
    nonReentrant
{
    // CHECKS
    if (!depositsEnabled) revert DepositsDisabled();
    if (address(birbOft) == address(0)) revert BirbOftNotConfigured();
    if (solanaAddress == bytes32(0)) revert InvalidSolanaAddress();

    bytes32 escrowKey = _getEscrowKey(collection, tokenId);
    if (escrows[escrowKey].active) revert AlreadyNested();
```

```
    uint256 tokenAmount = _getTokenAmount(collection);
    address tokenAddr = birbOft.token();
```

2. Redemption methods use current rate, not rate at nesting time.

```
    function lzCompose(
...
    ) external payable override whenNotPaused nonReentrant {
...
        uint256 requiredAmount = _getTokenAmount(collection)
    + redemptionFee;
        if (amountReceived < requiredAmount)
    revert InsufficientTokensReceived();
```

Although the owner controls the timing of the rate lock, since owner is also expected to tune rate, this creates edge cases of user fund loss: Case1: Rate lock after some user already nested The early users would receive a different amount of tokens on solana chain compared to later users nested after rate lock for the same collection. Since the collection would require the same token price at redemption(`lzCompose`/`releaseWithTokens`) after rate lock, this creates unfair token distributions.

Case2: Rate lock after some user already initiated redemption from solana chain. The new rate would be used to check against bridge funds. If the new rate decreases, users would have extra funds locked in the adapter contract.

Recommendations:

In `nest()`, consider adding a check to only allow nesting when `tokenRates.locked == true`.

Moonbirds: Acknowledged. We require the flexibility to adjust rates post-launch if necessary. The depositsEnabled flag provides sufficient manual control — deposits can be disabled before any rate changes to prevent gaming. This is an intentional design decision to maintain operational flexibility.

[L-2] nest() DoS due to OFT fee charges or rounding logic

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

Target

- MoonbirdsEscrowAdapter.sol#L203
- MoonbirdsEscrowAdapter.sol#L466

Description:

The `nest()` and `quoteNest()` functions always set `minAmountLD = tokenAmount` when constructing OFT send parameters. This creates a DoS vulnerability because the actual amount received (`amountReceivedLD`) can be less than `tokenAmount` due to OFT fee charges or dust removal (decimal conversion rounding), causing legitimate transactions to revert with `SlippageExceeded`.

```
// nest()
SendParam memory sendParam = SendParam({
    dstEid: solanaEid,
    to: solanaAddress,
    amountLD: tokenAmount,
▷   minAmountLD: tokenAmount,
    extraOptions: extraOptions,
    composeMsg: "",
    oftCmd: ""
});
// quoteNest()
SendParam memory sendParam = SendParam({
    dstEid: solanaEid,
    to: solanaAddress,
    amountLD: tokenAmount,
▷   minAmountLD: tokenAmount,
    extraOptions: extraOptions,
    composeMsg: "",
    oftCmd: ""
});
```

The protocol currently uses the `redemptionFee` model. However, if BirbOFT fees can be

enabled in the future, it will reduce`amountReceivedLD` below `amountSentLD`, causing the slippage check against `minAmountLD`to fail.

Recommendations:

1. In `nest()` and `quoteNest`, consider allowing the user to specify `minAmountLD` through input.
2. Or consider adding documentation to clarify there are no intended BirbOFT fees.

Moonbirds: Acknowledged. BirbOFT will never charge transfer fees. This is by design and is a deployment constraint we control.

[L-3] `nest` forwards user-supplied LayerZero `extraOptions` without enforcement, risking underfunded destination execution

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

Target

- src/MoonbirdsEscrowAdapter.sol#L160-L214

Description:

In `nest`, the adapter accepts arbitrary `extraOptions` from the caller and forwards it directly into `SendParam.extraOptions` for `birbOft.send()`. There is no validation that `extraOptions` contains the required executor options (e.g., sufficient gas/compose settings) for the destination chain.

If `extraOptions` is empty or underfunded/malformed, the transaction can still succeed on Ethereum and the adapter will:

- mark the NFT as escrowed (`escrows[escrowKey].active = true`), and
- transfer the NFT into the contract,

but the cross-chain delivery on Solana may fail, be delayed, or require a retry/executor intervention. This creates a user-impact scenario where the NFT is locked in escrow while the user does not receive the expected BIRB on Solana within normal flow.

This is primarily an operational/integration pitfall: correctness depends on callers always providing adequate `extraOptions` and sufficient `msg.value` to cover destination execution costs.

Recommendations:

It is recommended to enforce minimum LayerZero options for `nest` (e.g., via enforced options/configuration or validating that `extraOptions` contains required executor gas settings) and/or provide a documented recovery path for underfunded messages so users are not stranded with NFTs escrowed when cross-chain execution fails.

Moonbirds: Acknowledged. Callers are responsible for providing adequate `extraOptions` for LayerZero gas. This is documented behavior and the frontend/SDK handles option construction appropriately.

[L-4] Deployer implicitly owns `onlyOwner` privileges while `_owner` only sets `admin` and endpoint delegate

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

Target

- src/MoonbirdsEscrowAdapter.sol#L131-L156

Description:

`MoonbirdsEscrowAdapter` inherits LayerZero's `OAppCore`, which itself inherits OpenZeppelin `Ownable`. Under OpenZeppelin v4.x, `Ownable`'s `owner()` defaults to the deployer at construction time.

In `MoonbirdsEscrowAdapter`'s constructor, the `_owner` parameter is passed to `OAppCore(_endpoint, _owner)`, but `OAppCore` uses this value only to call `endpoint.setDelegate(_owner)`. The adapter then sets `admin = _owner`. The contract never transfers `Ownable` ownership to `_owner`.

As a result, there are two independent privileged identities:

- `owner()` (controls `onlyOwner`): the deployer address
- `admin` (controls `adminUnlock`/`adminUnlockBatch`): the `_owner` constructor argument

This can be intentional, but it is not explicitly documented and can cause misconfiguration in real deployments where the deployer is a factory, a script runner, or an EOA different from the intended governance/multisig. In such cases, critical controls (e.g., `pause`/`unpause`, `setPeer`, one-time setters like `setBirbOft` and `setRedemptionFee`, rate configuration/locking, deposits toggle) may be held by an unintended address, potentially resulting in loss of administrative control or incorrect assumptions about who can perform sensitive actions.

Recommendations:

It is recommended to explicitly document the intended privilege model and enforce it in tests by adding a case where `deployer ≠ _owner` and asserting the expected access control boundaries (which address can call `onlyOwner` functions vs which can call `adminUnlock*`).

Moonbirds: Resolved with `@a8e491f ...`

Zenith: Verified.

4.3 Informational

A total of 5 informational findings were identified.

[I-1] For-loops can be optimized to save gas in releaseWithTokensBatch

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

Target

- MoonbirdsEscrowAdapter.sol#L472-L484

Description:

The `releaseWithTokensBatch` method uses three for-loops that iterate over the same sequence of collection/token pairs. The first two loops (validation and token calculation) can be combined to reduce gas costs without changing functionality.

1. First loop: Validate escrow records.

```
for (uint256 i = 0; i < tokenIds.length; i++) {
    bytes32 escrowKey = _getEscrowKey(collections[i], tokenIds[i]);
    EscrowRecord storage record = escrows[escrowKey];
    if (!record.active) revert NotNested();
    if (msg.sender ≠ record.owner) revert NotOriginalOwner();
}
```

2. Second loop: Calculate total tokens.

```
uint256 totalTokens = 0;
for (uint256 i = 0; i < collections.length; i++) {
    totalTokens += _getTokenAmount(collections[i]);
}
```

3. Third loop: state changes.

Impacts: Unnecessary gas consumption from redundant loop overhead. Gas savings scale with batch size (approximately 50 gas per item).

Recommendations:

Consider combining the validation and token calculation loops into a single loop.

Moonbirds: Resolved with `@a8e491f ...`

Zenith: Verified.

[I-2] Unused import

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

Target

- MoonbirdsEscrowAdapter.sol#L5

Description:

The `MoonbirdsEscrowAdapter` contract contains an unused `Ownable` import. The contract already inherits `Ownable` from the parent contract OAppCore.

```
import {Ownable} from "@openzeppelin/contracts/access/Ownable.sol";
```

Recommendations:

Consider removing unused code.

Moonbirds: Resolved with `@a8e491f ...`.

Zenith: Verified.

[I-3] `depositsEnabled` defaults to `true` and is not tied to rate finalization

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

Target

- src/MoonbirdsEscrowAdapter.sol#L79

Description:

`MoonbirdsEscrowAdapter` initializes `depositsEnabled` to `true`, enabling `nest` immediately after deployment (subject to `birbOft` being configured and the adapter being funded). The ability to nest is not coupled to rate finalization (`lockRates()`), meaning users may be able to nest while `tokenRates` are still mutable via `setTokenRates()`.

Because redemption requirements are derived from current `tokenRates` at redemption time, allowing nesting before rates are finalized can create inconsistent user outcomes and operational risk during launch/configuration windows (e.g., rates changing after some users have already nested).

Recommendations:

It is recommended to default `depositsEnabled` to `false` and only enable deposits after completing launch-critical steps, including setting `birbOft`, configuring peers/pathway security, funding the adapter, and finalizing token economics by calling `lockRates()` (and setting `setRedemptionFee()` if applicable), so nesting cannot occur until rates are finalized.

Moonbirds: Resolved with `@ec068d6 ...`

Zenith: Verified.

[I-4] `nest` uses `msg.sender` as the LayerZero refund address

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

Target

- src/MoonbirdsEscrowAdapter.sol#L211

Description:

In `nest`, the adapter calls `birbOft.send( ... )` and sets the refund address to `payable(msg.sender)`:

- `birbOft.send{value: msg.value}(sendParam, fee, payable(msg.sender));`

In LayerZero/OFT flows, the refund address is typically used to return any excess native fee (and/or handle refund semantics depending on configuration). If `nest` is invoked through a smart contract (e.g., a relayer, router or account abstraction wallet), `msg.sender` will be that intermediary contract rather than the end user.

As a result, any fee refunds may be sent to the intermediary instead of the intended user, which can create unexpected user-facing discrepancies or require additional handling by the calling contract.

Recommendations:

It is recommended to explicitly document refund address behavior for integrators, or consider allowing the caller to specify a refund address.

Moonbirds: Acknowledged. Users calling via a contract should be aware that LayerZero refunds go to `msg.sender` (the calling contract). This is standard LayerZero behavior and is documented.

[I-5] `lzCompose` failures can leave NFTs escrowed and funds stranded

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

Target

- src/MoonbirdsEscrowAdapter.sol#L216-L272

Description:

LayerZero v2 OFT "compose" delivery is a multi-step process on the destination chain: the OFT transfer/receive occurs, and the composed call (`lzCompose`) is executed later (and may be retried). These steps are not guaranteed to be atomic.

In `MoonbirdsEscrowAdapter`, releasing an escrowed NFT via the cross-chain path depends entirely on successful execution of `lzCompose`. If `lzCompose` reverts for any reason (e.g., insufficient gas/options, paused state, malformed payload, `InsufficientTokensReceived`, fee transfer failure, or NFT transfer failure), the NFT remains escrowed (`record.active` stays `true`), and the redeem operation is incomplete.

Because token delivery and compose execution can be separated, a failure in `lzCompose` can result in a "partial completion" state where the required tokens may have already arrived to the adapter/OFT on the destination chain, but the NFT remains locked until compose is successfully retried or an alternative recovery action is taken. The contract does not provide a dedicated user-facing "retry/claim redeem" function; recovery is implicitly dependent on LayerZero executor retries or privileged/unrelated paths (e.g., `adminUnlock*` or `releaseWithTokens`).

Recommendations:

It is recommended to document the expected compose retry behavior and operational runbook, and to provide (or explicitly rely on) a clear recovery mechanism for cases where compose execution fails (e.g., a deterministic retry/claim path or explicitly defined admin intervention policy) so users are not stranded with NFTs escrowed when cross-chain compose execution does not complete.

Moonbirds: Acknowleged. Recovery paths exist for stranded NFTs:

1. `adminUnlock` - Admin can return NFT to original owner

2. `releaseWithTokens` - User can redeem directly on Ethereum after compose retry/failure

These mechanisms ensure NFTs are never permanently stuck.