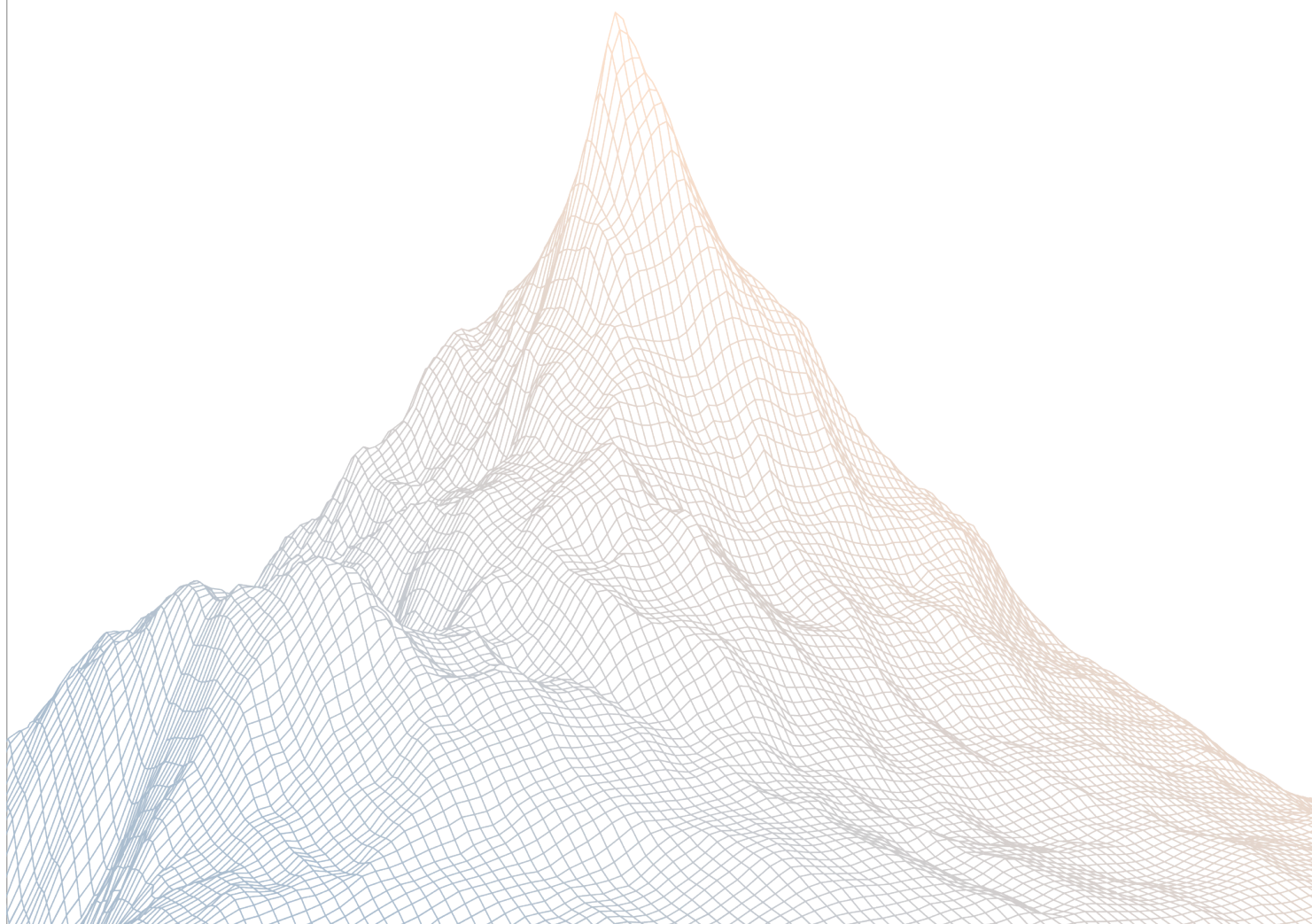


# Moonbirds

## Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: January 20th to January 23rd, 2026  
AUDITED BY: Mario Ponder  
oakcobalt

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	About Zenith	3
1.2	Disclaimer	3
1.3	Risk Classification	3
<hr/>		
<b>2</b>	<b>Executive Summary</b>	<b>3</b>
2.1	About Moonbirds	4
2.2	Scope	4
2.3	Audit Timeline	5
2.4	Issues Found	5
<hr/>		
<b>3</b>	<b>Findings Summary</b>	<b>5</b>
<hr/>		
<b>4</b>	<b>Findings</b>	<b>6</b>
4.1	Critical Risk	7
4.2	Medium Risk	9
4.3	Low Risk	19
4.4	Informational	23

# 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:

<b>Target</b>	bird-distributor
---------------	------------------

<b>Repository</b>	<a href="https://github.com/Moonbirds-OCG/bird-distributor.git">https://github.com/Moonbirds-OCG/bird-distributor.git</a>
-------------------	---

<b>Commit Hash</b>	644a522786aa40e2de135216f037d67f749446fd
--------------------	--

<b>Files</b>	programs/birb-distributor/src/**/*.rs
--------------	---------------------------------------

<b>Target</b>	birb-solana-oft
---------------	-----------------

<b>Repository</b>	<a href="https://github.com/Moonbirds-OCG/birb-solana-oft.git">https://github.com/Moonbirds-OCG/birb-solana-oft.git</a>
-------------------	---

<b>Commit Hash</b>	c00dcfb102caac0ba266c933df7deaedace6ad36
--------------------	--

<b>Files</b>	programs/oft/src/**/*.rs
--------------	--------------------------

## 2.3 Audit Timeline

<b>January 20, 2026</b>	Audit start
<b>January 23, 2026</b>	Audit end
<b>January 26, 2026</b>	Report published

## 2.4 Issues Found

SEVERITY	COUNT
Critical Risk	1
High Risk	0
Medium Risk	6
Low Risk	2
Informational	5
<b>Total Issues</b>	<b>14</b>

## 3

## Findings Summary

ID	Description	Status
C-1	Incorrect secp256k1 instruction introspection allows signature-message mismatch	Resolved
M-1	quote_send encodes LD as SD in message payload, causing fee quotes to mismatch send	Resolved
M-2	Token vault ATA front-run initialization DOS	Resolved
M-3	amount_sd conversion can bypass slippage check	Resolved
M-4	Rate limiter capacity can be bypassed due to vulnerable capacity update	Resolved
M-5	batch_claim PDA creation is vulnerable to prefund grieving that permanently blocks claims for targeted ETH addresses	Resolved
M-6	Iz_receive enforces native mint_authority constraint even in Adapter mode, causing valid Adapter receives to fail	Resolved
L-1	Panic-based DoS risks from unchecked indexing in message parsing and remaining_accounts handling	Resolved
L-2	Claim signature lacks domain separation / nonce / expiry, enabling replay in other contexts that accept the same EIP-191 message	Resolved
I-1	Missing underflow check in Iz_receive	Resolved
I-2	Zero-amount claims allowed, enabling exhaustion of max_num_nodes without distributing tokens	Resolved
I-3	Overly strict requirement that secp256k1 instruction is at index 0 reduces composability	Resolved
I-4	Missing length checks can cause panic	Resolved
I-5	OFTStore PDA is derived from token_escrow address, allowing multiple instances per mint	Acknowledged

# 4

## Findings

### 4.1 Critical Risk

A total of 1 critical risk findings were identified.

#### [C-1] Incorrect secp256k1 instruction introspection allows signature-message mismatch

SEVERITY: Critical

IMPACT: High

STATUS: Resolved

LIKELIHOOD: High

#### Target

- [programs/birb-distributor/src/instructions/claim.rs#L171-L249](https://github.com/birb-distributor/src/instructions/claim.rs#L171-L249)

#### Description:

The `verify_secp256k1_signature` helper in `claim.rs` parses the `secp256k1` instruction data but ignores the `*_instruction_index` fields (`signature_instruction_index`, `eth_address_instruction_index`, `message_instruction_index`). It then unconditionally reads the Ethereum address and message bytes from the data of instruction index 0:

- The code assumes that all three indices are 0 and that the secp program reads signature, address, and message from the same instruction whose data it inspects.
- However, the `secp256k1` program actually uses these indices to select which instruction's data to read for each component.

Because the on-chain code never validates that these indices are 0, an attacker can:

- Craft a `secp256k1` instruction where the actual verified message comes from a different instruction (`message_instruction_index`  $\neq$  0), containing some arbitrary message that they truly signed.
- Populate the bytes at `eth_address_offset` and `message_offset` inside instruction 0's data with:

- An Ethereum address equal to `expected_eth_address`, and
- A message that looks exactly like the expected EIP-191 prefixed string

```
"\x19Ethereum Signed Message:\n<len>Claim BIRB tokens to Solana address:
<expected_claimant>".
```

- The secp program will successfully verify the signature over the other message, while `verify_secp256k1_signature` will incorrectly conclude that:

- The recovered Ethereum address matches `expected_eth_address`, and
- The message bytes match the expected claim message for `expected_claimant`.

This breaks the intended guarantee that “the ETH address holder has signed the specific claim message for this Solana recipient”, and opens the door to signature replay/substitution, where a valid signature over one message can be reused to authorize a different claim as long as the attacker can shape instruction O’s data accordingly.

### Recommendations:

It is recommended to fully validate the `secp256k1` instruction layout, including all instruction-index fields, before trusting the recovered address and message, specifically:

- Parse and enforce that `signature_instruction_index`, `eth_address_instruction_index`, and `message_instruction_index` are all 0, so that:
  - The secp program reads the signature, ETH address, and message from the same instruction whose data is being introspected.
- Optionally add extra consistency checks (e.g. that the layout after the 12-byte header matches `signature(64) + recovery_id(1) + eth_address(20) + message( ... )`), to make it harder to craft misleading instruction data.

**Moonbirds:** Resolved with [@4c2e702 ...](#)

**Zenith:** Verified.

## 4.2 Medium Risk

A total of 6 medium risk findings were identified.

[M-1] `quote_send` encodes LD as SD in message payload, causing fee quotes to mismatch `send`

SEVERITY: Medium

IMPACT: Medium

STATUS: Resolved

LIKELIHOOD: Medium

### Target

- [programs/oft/src/instructions/quote\\_send.rs#L58](#)

### Description:

The message codec `msg_codec::encode(send_to, amount_sd, ...)` stores the amount field in SD (shared decimals). The `send` instruction correctly converts `amount_received_ld` to SD via `oft_store.ld2sd(amount_received_ld)` before encoding the message. However, `quote_send` passes `amount_received_ld` directly into `msg_codec::encode(...)` as if it were SD.

This results in `quote_send` generating a different payload than `send` for the same user parameters. For typical mints where `ld2sd_rate = 10^(local_decimals - shared_decimals) > 1`, the encoded amount in `quote_send` is inflated by a factor of `ld2sd_rate`, which can lead to incorrect fee quotes and integration issues because `quote_send` no longer quotes the exact message that will be sent by `send`.

### Recommendations:

It is recommended to convert `amount_received_ld` to SD inside `quote_send` (the same way `send` does) and pass that SD value to `msg_codec::encode(...)`, ensuring `quote_send` quotes fees for the exact payload `send` will submit.

**Moonbirds:** Resolved with [@c4f7cee...](#), and test added with [@947fbb1...](#)

**Zenith:** Verified.

## [M-2] Token vault ATA front-run initialization DOS

SEVERITY: Medium

IMPACT: Medium

STATUS: Resolved

LIKELIHOOD: Low

### Target

- [src/instructions/initialize.rs#L26-L32](#)

### Description:

The token\_vault account in initialize.rs uses init constraint for an ATA account.

This allows an attacker to preemptively initialize the token\_vault associated token account before the admin's initialization transaction, causing initialize\_distributor instruction to fail.

They can calculate the address and initialize the ATA with the following code after the program is deployed, before Initialize is called. For example,

```
// Attacker derives distributor PDA (seeds are public)
const [distributorPDA] = PublicKey.findProgramAddressSync(
  [Buffer.from("MerkleDistributor"), mintAddress.toBuffer()],
  program.programId
);

// Attacker calculates token_vault ATA address
const tokenVaultATA = await getAssociatedTokenAddress(
  mintAddress,
  distributorPDA, // owner (PDA)
  true,          // allowOwnerOffCurve
  TOKEN_PROGRAM_ID
);
const tx = new Transaction().add(
  createAssociatedTokenAccountInstruction(
    attackerKeypair.publicKey, // payer
    tokenVaultATA,             // ATA address
    distributorPDA,             // owner
    mintAddress
  )
);
await sendAndConfirmTransaction(connection, tx, [attackerKeypair]);
```

Impacts: This allows a DOS attack on the initialization of any distributor, particularly vulnerable to future mint-distributor pair creations, preventing potential scaling to support additional tokens.

### **Recommendations:**

Consider using `init_if_needed` instead of `init` for `token_vault`.

**Moonbirds:** Resolved with [@586f136 ...](#)

**Zenith:** Verified.

## [M-3] amount\_sd conversion can bypass slippage check

SEVERITY: Medium

IMPACT: Medium

STATUS: Resolved

LIKELIHOOD: Medium

### Target

- [instructions/send.rs#L65](#)

### Description:

The slippage check in `send.rs` validates `amount_received_ld` (local decimals) but does not account for conversion to `amount_sd` (shared decimals) that occurs later. There is no additional check on converted `amount_sd`.

```
require!(  
    amount_received_ld ≥ params.min_amount_ld,  
    OFTErrors::SlippageExceeded  
);
```

However, the actual amount sent cross-chain is `amount_sd = ld2sd(amount_received_ld)`, which is calculated later through integer division (`ld2sd()` → `amount_ld / self.ld2sd_rate`). This creates risks of fund loss depending on mint token values and local/share decimal conversions. For example, Suppose a token with 8 decimals, shared decimals is 4, Token price = \$50,000. `ld2sd_rate = 10^(8-4) = 10,000`.

1. User sets `min_amount_ld = 9,999`. After fee: `amount_received_ld = 9,999`
2. Slippage check passes.
3. `amount_sd = ld2sd(9,999) = 0`, leading to around \$5 loss.

Impact: Medium - Low The impact is conditional on `ld2sd_rate` configuration and token value.

Likelihood: Medium Permissionless deployment of `off_store` with custom config is allowed.

### Recommendations:

Consider checking slippage for `amount_sd` or check `amount_sd > 0`.

**Moonbirds:** Resolved with [@c4f7cee ...](#)

**Zenith:** Verified.

## [M-4] Rate limiter capacity can be bypassed due to vulnerable capacity update

SEVERITY: Medium

IMPACT: Medium

STATUS: Resolved

LIKELIHOOD: Medium

### Target

- [peer\\_config.rs#L34](#)

### Description:

When an admin updates a rate limiter's capacity via `set_capacity()`, tokens are immediately reset to the new capacity, allowing an attacker who back-runs the admin's transaction to bypass rate limits by consuming both the old capacity and the newly set capacity in the same block or subsequent block.

In `set_capacity()`, `tokens` is always set to `capacity` when updated, regardless of current token state. This breaks the rate limiting guarantee that after initialization, tokens should refill gradually over time based on `refill_per_second`.

```
pub fn set_capacity(&mut self, capacity: u64) → Result<()> {
    self.capacity = capacity;
    self.tokens = capacity;
    self.last_refill_time
    = Clock::get()?.unix_timestamp.try_into().unwrap();
    Ok(())
}
```

Suppose the following case:

1. Initial state: `capacity = 1000`, `tokens = 1000`
2. Attacker sends 1000 tokens → `tokens = 0` (rate limit consumed)
3. Admin calls `set_peer_config` with `OutboundRateLimit` to reduce capacity to 100.
4. `set_capacity(100)` executes → `tokens = 100`
5. Attacker back-runs with another send transaction consuming 100 tokens.
6. Result: Attacker sent  $1000 + 100 = 1100$  tokens in the same/subsequent block, exceeding both old and new capacity.

Impacts: Medium Allows bypassing rate limits when an admin updates capacity, especially dangerous when the admin is restricting capacity.

Likelihood: Medium Permissionless method triggers the attack, but it depends on the timing of the admin action.

### Recommendations:

Considering updating `set_capacity()` to distinguish between initialization and capacity updates. For example,

1. Initialization (`old_capacity == 0`). Set tokens = capacity (immediately usable)
2. Capacity update (`old_capacity > 0`): Only update capacity and cap tokens at the new capacity.

**Moonbirds:** Resolved with [@d22c137 ...](#)

**Zenith:** Verified.

[M-5] `batch_claim` PDA creation is vulnerable to prefund grieving that permanently blocks claims for targeted ETH addresses

SEVERITY: Medium

IMPACT: Medium

STATUS: Resolved

LIKELIHOOD: Medium

## Target

- [programs/birb-distributor/src/instructions/batch\\_claim.rs#L142-L186](#)

## Description:

In `programs/birb-distributor/src/instructions/batch_claim.rs`, each claim in the batch creates a `ClaimStatus` PDA using:

```
let (expected_pda, bump) = Pubkey::find_program_address(
    &[
        ClaimStatus::SEED_PREFIX,
        distributor.key().as_ref(),
        &claim_data.eth_address,
    ],
    ctx.program_id,
);
require!(
    claim_status_info.key() == expected_pda,
    DistributorError::InvalidClaimStatusPDA
);
```

Because PDAs are deterministic and publicly derivable, an attacker can compute the `expected_pda` for any ETH address in the Merkle tree (given the distributor pubkey and program id).

The code checks only that the account data is empty:

```
require!(
    claim_status_info.data_is_empty(),
    DistributorError::AlreadyClaimed
);
```

It then creates the PDA using `system_instruction::create_account` via `invoke_signed`:

```
invoke_signed(  
    &system_instruction::create_account(  
        &ctx.accounts.claimant.key(),  
        &expected_pda,  
        lamports,  
        space as u64,  
        ctx.program_id,  
    ),  
    // ...  
)?;
```

This creation method is susceptible to a griefing attack: if an attacker transfers a small amount of SOL to `expected_pda` before a legitimate claim occurs, the PDA address can become an already-existing system-owned account with lamports. The System Program `create_account` implementation rejects creation when the destination account's lamports are non-zero (i.e., "already in use"), causing the `batch_claim` initialization to fail.

### Recommendations:

It is recommended to use a more resilient initialization flow (e.g., allocate + assign + transfer pattern, see also source code of Anchor's `init`) that can take an existing system-owned, zero-data account at the PDA address and convert it into a program-owned account of the required size.

**Moonbirds:** Resolved with [@d6460ee ...](#)

**Zenith:** Verified.

## [M-6] lz\_receive enforces native mint\_authority constraint even in Adapter mode, causing valid Adapter receives to fail

SEVERITY: Medium

IMPACT: Medium

STATUS: Resolved

LIKELIHOOD: Medium

### Target

- [programs/oft/src/instructions/lz\\_receive.rs#L65-L66](#)

### Description:

In `programs/oft/src/instructions/lz_receive.rs`, the `mint_authority` account is declared as optional (`Option<AccountInfo<'info>>`) and documented as “Only used for native mint”. However, it has an unconditional Anchor account constraint:

```
#[account(constraint = token_mint.mint_authority
    = COption::Some(mint_authority.key()) @OFTErrror::InvalidMintAuthority)]
pub mint_authority: Option<AccountInfo<'info>>,
```

Anchor evaluates this constraint during account validation before the instruction handler runs. As a result, even when `oft_store.oft_type = OFTType::Adapter` (where the handler does not mint and does not need a mint authority), the transaction can still fail due to the native mint authority constraint being enforced.

This can break Adapter-mode `lz_receive` flows for tokens/mints whose `mint_authority` is unset or does not match the provided `mint_authority` account, despite Adapter mode only unlocking from escrow and not minting.

### Recommendations:

It is recommended to gate the `mint_authority` constraint, so it is enforced only when `oft_store.oft_type = Native`.

**Moonbirds:** Resolved with [@c4f7cee ...](#)

**Zenith:** Verified.

## 4.3 Low Risk

A total of 2 low risk findings were identified.

### [L-1] Panic-based DoS risks from unchecked indexing in message parsing and remaining\_accounts handling

SEVERITY: Low

IMPACT: Low

STATUS: Resolved

LIKELIHOOD: Medium

#### Target

- [programs/oft/src/instructions/send.rs#L831](#)

#### Description:

Several instruction paths can panic due to unchecked indexing/slicing, instead of failing with a controlled program error. This is most concerning for externally-driven flows such as `lz_receive`, where the message bytes originate from a remote sender and may be malformed.

Affected patterns include:

- Unchecked slice indexing in message decoding (`msg_codec`):
  - `send_to( ... )` slices `message[0..32]`
  - `amount_sd( ... )` slices `message[32..40]`
  - `compose_msg( ... )` slices `message[40..]`
  - If `params.message` is shorter than expected, these can panic.
- Unchecked `remaining_accounts` indexing/slicing:
  - `send` accesses `ctx.remaining_accounts[1]` without verifying length.
  - `lz_receive` slices `ctx.remaining_accounts[0..Clear::MIN_ACCOUNTS_LEN]` without verifying length.

While a panic still causes the transaction to fail, panics in receive/executor flows can enable griefing/DoS by causing repeated delivery attempts to abort (and they are harder to classify/handle than explicit, typed errors).

**Recommendations:**

It is recommended to add explicit length checks and use safe parsing helpers so malformed `params.message` and insufficient `remaining_accounts` return a deterministic program error (e.g., `InvalidMessage` / `InvalidRemainingAccounts`) rather than panicking. This should be applied particularly to `lz_receive` (remote-controlled inputs) and any place that indexes into `remaining_accounts` or slices raw message bytes.

**Moonbirds:** Resolved with [@c4f7cee ...](#)

**Zenith:** Verified.

[L-2] Claim signature lacks domain separation / nonce / expiry, enabling replay in other contexts that accept the same EIP-191 message

SEVERITY: Low

IMPACT: Medium

STATUS: Resolved

LIKELIHOOD: Low

## Target

- [programs/birb-distributor/src/instructions/claim.rs#L237-L241](#)

## Description:

In `programs/birb-distributor/src/instructions/claim.rs`, ETH ownership is proven by requiring a `secp256k1`-verified signature over an EIP-191 “Ethereum Signed Message” whose base message is:

- "Claim BIRB tokens to Solana address: <solana\_pubkey>"

The program reconstructs this exact expected message and compares it byte-for-byte:

```
let base_message = format!("Claim BIRB tokens to Solana address: {}",
    expected_claimant);
let prefix = format!("\x19Ethereum Signed Message:\n{}",
    base_message.len());
let expected_message = format!("{}", prefix, base_message);

require!(
    message == expected_message.as_bytes(),
    DistributorError::SignatureVerificationFailed
);
```

This message binds the signature to a Solana recipient pubkey, which prevents simple front-running where an attacker tries to redirect the claim to their own Solana address.

However, the signed content does not include any of the following:

- program id / distributor id (PDA) / mint
- network identifier (e.g., Solana cluster or an Ethereum chain id)
- a nonce, per-claim unique value, or expiration timestamp

Because of that, the signature is not domain-separated to this specific program/distributor and can function as a standing authorization: any other system (another Solana program, off-chain service, or even a contract/app on another chain) that treats “a valid EIP-191 signature over this exact string” as authorization could accept and reuse it.

*(Within this program specifically, replay is limited by the per-ETH-address `claimStatus` PDA preventing double-claims, but the lack of domain separation still matters outside this program.)*

## Recommendations:

It is recommended to consider the following mitigation measures:

- Add domain separation into the signed message, such as including:
  - the program id and/or distributor PDA,
  - the mint,
  - and an explicit environment/cluster identifier.
- Add a nonce and/or expiry:
  - e.g., include a distributor-specific nonce, a per-claim random nonce, and/or an expiration timestamp, and enforce it on-chain.
- Consider switching to a structured signing scheme (e.g., EIP-712-like structured data off-chain), while still verifying on-chain via `secp256k1`, to reduce ambiguity and improve wallet UX/security.

**Moonbirds:** Resolved with [@586f136 ...](#)

**Zenith:** Verified. Resolved by including the distributor pubkey in the message.

## 4.4 Informational

A total of 5 informational findings were identified.

### [I-1] Missing underflow check in lz\_receive

SEVERITY: Informational

IMPACT: Informational

STATUS: Resolved

LIKELIHOOD: Low

#### Target

- [src/instructions/lz\\_receive.rs#L111](#)

#### Description:

The `lz_receive` instruction uses direct subtraction for `tv1_ld` decrement instead of `checked_sub`, which can panic on underflow.

```
ctx.accounts.oft_store.tv1_ld -= amount_received_ld;
```

Although the current token Oft development config uses a native type, which doesn't touch `tv1_ld`, an underflow panic case can occur in an adapter type Oft receive transaction deployed through the program.

#### Recommendations:

Consider using `checked_sub` with error handling.

**Moonbirds:** Resolved with [@c4f7cee ...](#)

**Zenith:** Verified.

## [I-2] Zero-amount claims allowed, enabling exhaustion of max\_num\_nodes without distributing tokens

SEVERITY: Informational

IMPACT: Informational

STATUS: Resolved

LIKELIHOOD: Low

### Target

- [programs/birb-distributor/src/instructions/claim.rs#L74](#)

### Description:

In programs/birb-distributor/src/instructions/claim.rs, the claim instruction accepts amount: u64 and does not require amount > 0 before updating distributor state.

The code always increments num\_claimed by 1 for every successful claim:

```
let new_num = distributor
    .num_claimed
    .checked_add(1)
    .ok_or(DistributorError::ArithmeticOverflow)?;
require!(
    new_num ≤ distributor.max_num_nodes,
    DistributorError::ExceedsMaxNumNodes
);
```

Because amount can be zero, a claimant with a valid Merkle proof for a leaf where amount = 0 (or any distribution that includes such leaves) can submit a successful claim that:

- creates a ClaimStatus account (consuming rent / state space),
- increments distributor.num\_claimed,
- transfers 0 tokens (a no-op at the token level, but still counts as a “claim” for node-limit purposes).

This creates a denial-of-service vector against the distribution’s intended availability: max\_num\_nodes can be consumed by zero-value claims, potentially preventing later legitimate claims once the node limit is reached.

**Recommendations:**

It is recommended to reject zero-amount claims by enforcing `require!(amount > 0, ...)` early in `claim` before state updates and CPI.

**Moonbirds:** Resolved with [@586f136 ...](#)

**Zenith:** Verified.

## [I-3] Overly strict requirement that secp256k1 instruction is at index 0 reduces composability

SEVERITY: Informational

IMPACT: Informational

STATUS: Resolved

LIKELIHOOD: Low

### Target

- [programs/birb-distributor/src/instructions/claim.rs#L181-L185](https://github.com/birb-distributor/src/instructions/claim.rs#L181-L185)

### Description:

In `programs/birb-distributor/src/instructions/claim.rs`, `verify_secp256k1_signature()` currently loads the `secp256k1` instruction strictly from transaction instruction index 0, and it checks only that the current instruction index is greater than 0 (`current_idx > 0`), and then unconditionally loads `ix[0]` as the `secp` instruction.

This means the program does not enforce the documented “claim is exactly at index 1” structure; it only enforces “claim is not index 0”, while still requiring that the `secp` instruction is at index 0.

As a result, any transaction where `claim` is at index 2+ (which is allowed today by `current_idx > 0`) will still be required to place the `secp` instruction at index 0, even if there are unrelated setup instructions that naturally precede it. This is unnecessarily restrictive and can break composability with common transaction patterns that insert instructions earlier in the transaction (e.g., compute budget / fee / tip setup), without providing additional security over a requirement that `secp` immediately precedes `claim`.

### Recommendations:

It is recommended to enforce that the `secp256k1` instruction is immediately preceding the `claim` instruction rather than forcing it to be at index 0:

- Load the `secp` instruction from `current_idx - 1` instead of hardcoding index 0, and require `current_idx > 0`.
- Keep verifying that the loaded instruction’s `program_id` is the `secp256k1` program.
- This aligns the check with what the code currently enforces about `current_idx` (it does not require `current_idx == 1`), while improving composability by allowing arbitrary instructions to appear before the `secp+claim` pair as long as the `secp` instruction directly precedes the `claim`.

**Moonbirds:** Resolved with [@586f136 ...](#)

**Zenith:** Verified.

## [I-4] Missing length checks can cause panic

SEVERITY: Informational

IMPACT: Informational

STATUS: Resolved

LIKELIHOOD: Low

### Target

- [src/instructions/set\\_peer\\_config.rs#L38-L40](#)
- [src/instructions/lz\\_receive\\_types.rs#L61](#)
- [src/instructions/lz\\_receive\\_types.rs#L113](#)

### Description:

Several functions call slice operations without validating input length, which can cause the program to panic if inputs are shorter than expected. For example:

#### 1. PeerConfigParam::EnforcedOptions

```
PeerConfigParam::EnforcedOptions {  
    send,  
    send_and_call,  
} ⇒ {  
    // @audit This panics when send and send_and_call < 2 bytes  
    oapp::options::assert_type_3(&send)?;  
    ctx.accounts.peer.enforced_options.send = send;  
    oapp::options::assert_type_3(&send_and_call)?;  
    ctx.accounts.peer.enforced_options.send_and_call = send_and_call;  
}
```

#### 2. lz\_receive\_types.rs

```
//@audit If `message.len() < 40`, `copy_from_slice` will panic with index  
out of bounds.  
let to_address = Pubkey::from(msg_codec::send_to(&params.message));  
...  
let amount_sd = msg_codec::amount_sd(&params.message);
```

**Recommendations:**

Consider adding explicit length validation on the above input variables with proper error handling.

**Moonbirds:** Resolved with [@d22c137 ...](#)

**Zenith:** Verified.

[I-5] OFTStore PDA is derived from token\_escrow address, allowing multiple instances per mint

SEVERITY: Informational

IMPACT: Informational

STATUS: Acknowledged

LIKELIHOOD: Low

### Target

- [programs/oft/src/instructions/init\\_oft.rs#L9-L16](#)

### Description:

In programs/oft/src/instructions/init\_oft.rs, the oft\_store PDA is derived using the newly created token\_escrow account's address:

```
#[account(
  init,
  payer = payer,
  space = 8 + OFTStore::INIT_SPACE,
  seeds = [OFT_SEED, token_escrow.key().as_ref()],
  bump
)]
pub oft_store: Account<'info, OFTStore>,
```

Because token\_escrow is initialized during init\_oft, this design permits creating multiple distinct OFTStore accounts for the same token\_mint (each with a different escrow address). This may be intended to support multiple independent OFT instances, but it is non-canonical if the intended model is “one OFT instance per mint,” and it can affect discoverability and integration assumptions.

### Recommendations:

It is recommended to clearly document whether the design intends to support multiple OFT instances per mint. If a single canonical instance per mint is desired, derive the OFTStore PDA from stable identifiers such as token\_mint (and then derive token\_escrow from the OFTStore), rather than seeding the store by a newly created escrow address.

**Moonbirds:** Acknowledged. This is a design decision. The OFTStore PDA derivation from token\_escrow allows flexibility for different deployment scenarios. The admin controls initialization and this does not pose a security risk in the intended usage pattern.