# Fjord Foundry Security Audit

Report Version 1.0

June 4, 2024

Conducted by **Hunter Security**:

**BountyHunt3r**, Senior Security Auditor
**byter0x1**, Senior Security Auditor

# Table of Contents

# 1 About Hunter Security

Hunter Security consists of multiple teams of leading smart contract security researchers. Having conducted over 100 security reviews and reported tens of live smart contract security vulnerabilities protecting over $1B of TVL, our team always strives to deliver top-quality security services to DeFi protocols. For security audit inquiries, you can reach out to us on Telegram or Twitter at *@georgehntr*.

# 2 Disclaimer

Audits are a time-, resource-, and expertise-bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to identify as many vulnerabilities as possible. Audits can reveal the presence of vulnerabilities **but cannot guarantee their absence**.

# 3 Risk classification

| Severity | Impact: High | Impact: Medium | Impact: Low |
|---|---|---|---|
| Likelihood: High | High | High | Medium |
| Likelihood: Medium | High | Medium | Low |
| Likelihood: Low | Medium | Low | Low |

## 3.1 Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - involves a small loss of funds or affects a core functionality of the protocol.
- **Low** - encompasses any unexpected behavior that is non-critical.

## 3.2 Likelihood

- **High** - a direct attack vector; the cost is relatively low compared to the potential loss of funds.
- **Medium** - only a conditionally incentivized attack vector, with a moderate likelihood.
- **Low** - involves too many or unlikely assumptions; offers little to no incentive.

## 3.3 Actions required by severity level

- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

# 4 Executive summary

Hunter Security was engaged by Fjord Foundry to review their smart contract protocol during the period from May 11, 2024, to May 24, 2024.

**Overview**

| | |
|---|---|
| Project Name | Fjord Foundry |
| Repository | https://github.com/marigoldlabs/labrys-contracts |
| Commit hash | 9723847de8b5e57c08db628d8ec94f159fc241cd |
| Resolution | b148a438e24018277d05243d24ccb642098fdf1f |
| Methods | Manual review & testing |

**Timeline**

| | | |
|---|---|---|
| - | May 11, 2024 | Audit kick-off |
| v0.1 | May 24, 2024 | Preliminary report |
| v1.0 | June 4, 2024 | Mitigation review |

**Scope**

| |
|---|
| packages/contracts/programs/fjord-lbp/src/* |

**Issues Found**

| | |
|---|---|
| High risk | 3 |
| Medium risk | 7 |
| Low risk | 2 |

# 5 Findings

## 5.1 High

### 5.1.1 A treasury fee recipient can steal the fees of other users

**Severity:** High

**Context:** liquidity_bootstrapping_pool.rs

**Description:** The *retrieve_valid_keys()* function is called by *close_pool()* to fetch the fee recipients addresses. The function takes in the *treasury.fee_recipients* as its *a* parameter value and *ctx.remaining_accounts* as the *b* value. The *ctx.remaining_accounts* value is a vector of *AccountInfo* objects passed by the called *close_pool()*, which can be anyone.

The function creates a hashmap of user ATAs to fee percentages then filters the values of *b* and creates a new vector containing *FeeRecipient* structs that contain addresses that were provided in *b* and matched an ATA in the created hashmap.

The only check performed by the function to validate if the configuration of fee recipient addresses is correct is an *a.len() != filtered_b.len()* check, which is insufficient.

As mentioned above, *ctx.remaining_accounts* is under the control of the caller, allowing them to supply arbitrary and duplicate addresses in this vector of accounts. Suppose we have the following fee recipient configuration (where R# is an address that maps to a fee percentage):

```
R1 -> 40%
R2 -> 20%
R3 -> 20%
R4 -> 10%
```

Then whoever closes the pool can supply the following in *ctx.remaining_accounts*:

```
[R1, R2, R2, R4]
[R1, R3, R3, R4]
```

And so on, stealing other recipients' funds.

**Recommendation:** Ensure that the final recipient arrays contain no duplicate addresses. It is also recommended reading the recipients using the treasury, instead of depending on user-supplied input.

**Resolution:** Resolved. An additional check has been implemented to account for duplicate recipient addresses.

### 5.1.2 Arithmetic underflow allows for share stealing

**Severity:** High

**Context:** src

**Description:** The program is vulnerable to under/overflow attacks in multiple locations. Although the program uses cargo *underflow-checks* flag, this only works in debug mode.

However, when using *cargo build-bpf* for deployment it compiles the code into release mode, meaning it will be vulnerable to arithmetic under/overflow.

For instance, when a user is selling shares for the asset, the function *_swap_shares_for_assets()* is called, and it attempts to remove shares from the user:

```
user_state_in_pool.purchased_shares -= shares_in
```

A malicious user can underflow the balance of their shares and drain the entire pool from the assets.

**Recommendation:** Use Rust checked math.

**Resolution:** Resolved. Share/asset calculations now utilize Rust's safe math arithmetic operations from the *safe_math* lib.

#### 5.1.3 Silent loss of claimable shares/assets if they are greater than current reserve

**Severity:** High

**Context:** redemption.rs

**Description:** The function *redeem* currently has the following logic during claiming:

```
if ctx.accounts.pool_share_token_account.amount < shares {
    ctx.accounts.pool_share_token_account.amount
} else {
    shares
}
```

In case the claimable shares is greater than the current reserve in the *pool_share_token_account*, the user will get what is left in the reserves. But in that case, the user will lose a number of shares equal to *shares - ctx.accounts.pool_share_token_account.amount*. Since this difference is not registered, there is no way for the user to claim that amount.

A similar behavior exists for users claiming their referral fee, in terms of assets:

```
if ctx.accounts.pool_asset_token_account.amount < assets {
    ctx.accounts.pool_asset_token_account.amount
} else {
    assets
}
```

**Recommendation:** Consider writing the difference to the *user_stats.total_purchased/total_referred*.

**Resolution:** Resolved. When the reserves are insufficient to cover the user's purchased shares and referred assets, they are now deducted from the user's purchased shares and referred assets.

### 5.2 Medium

#### 5.2.1 Inconsistent design between the EVM and SVM versions

**Severity:** Medium

**Context:** liquidity_bootstrap_lib.rs

**Description:** Since the code is adapted from an EVM version, it is expected that the arithmetic operations be the same. However, when calculating the reserve in the EVM version the following happens:

- *assets = assets + virtualAssets* but in the SVM version we have *assets - virtualAssets*. This can also cause underflow errors in case the *assets* are 0, and only *virtualAssets* are used in the pool

**Recommendation:** Consider adapting this design to match the EVM version.

**Resolution:** Resolved. *assets = assets - virtualAssets* has been fixed to *assets = assets + virtualAssets*.

#### 5.2.2 Protocol does not consider the asset_swap_fees when calculating the exchange rate.

**Severity:** Medium

**Context:** liquidity_bootstrap_lib.rs

**Description:** Currently, the protocol does not remove the *total_swap_fees_asset* from the *asset_reserves* in the *compute_reserves_and_weights()*. This value is used later to calculate the exchange rate.

This can result in an inaccurate exchange rate, and potentially a loss for the protocol since the assets are less valuable than they are.

The same goes for the *share_reserves*, where the *share_swap_fees* should also be removed to have an accurate state.

**Recommendation:** When fetching the asset amount, subtract the *total_swap_fees_asset*.

**Resolution:** Resolved. Compute reserves and weights function now takes fees into account.

#### 5.2.3 Swapping a large amount of assets into shares can cause an overflow

**Severity:** Medium

**Context:** shared.rs

**Description:** Currently, swapping more than *30%* of the reserves is not allowed. However, when attempting to swap a large amount of shares, but that is less than *30%*, the execution will lead to a multiplication overflow.

Specifically, assuming that the reserves for the shares and assets are both *100_000_000*, with shares having 6 decimals and assets having 9 decimals. The maximum swap amount is around *30_300_000* approximately. However, attempting to swap assets that are worth *30_000_000* shares will throw an overflow error.

This happens in the *_scale_token* function because the *scaled_amount* is already near the bounds of *u64*, and then when the decimal scaling is applied, it overflows.

**Recommendation:** Because of the need for fitting the final amounts in *u64*, overflows are edge cases that can happen with a big enough value. It is recommended to be aware of this state, and document it for users.

**Resolution:** Acknowledged.

#### 5.2.4 A buyer can set themselves as the referrer and get a cashback on their buys

**Severity:** Medium

**Context:** buy.rs

**Description:** The *_swap_assets_for_shares()* function used by the two buy swap functions accepts a *_referrer_state_in_pool* account passed by the buyer, calculates the referral fee, and accumulates the fee balance collected by the referrer in the account.

Validation that the buyer is not the same as the referrer is not implemented, allowing buyers to buy shares at a discounted price.

**Recommendation:** Add a constraint that reverts if the keys of *user_state_in_pool* and *referrer_state_in_pool* are equal.

**Resolution:** Acknowledged. Fjord's team is aware of this issue in both the EVM and SVM logic. Having a check to ensure the buyer is not the same as the referrer still does not prevent sybil issues, so Fjord has opted to disable the referral feature by setting the referral fee to 0.

#### 5.2.5 The referral fee can be smaller than the swap fee

**Severity:** Medium

**Context:** global_pool_fees.rs

**Description:** The *set_fees()* function does not validate that the referral fee percentage is smaller than that of the swap fee. This allows users to specify referrer addresses that they control (or just use the issue above) to get back their swap fees plus any surplus in case the referral fee percentage is bigger than the swap fee percentage.

For example, consider the following attack scenario:

1. Token A trades at 1:1 with token B (assets vs shares)
2. The swap fee is 5% and the referral fee is 10%
3. The user sends in 100 A tokens and their own address as the referrer (or another address they control)
4. The total they pay in A tokens is (100 + 5 - 10) = 95 tokens
5. The total B tokens they get is 100, so B tokens were 5% cheaper for them

**Recommendation:** Ensure that the referral fee percentage is smaller than that of the swap fee.

**Resolution:** Acknowledged. Fjord's team is aware of this issue, and has opted to disable the feature altogether by setting the referral fee to 0.

#### 5.2.6 Possible case for assets_in to round to 0 in preview_shares_in, and preview_assets_in

**Severity:** Medium

**Context:** liquidity_bootstrap_lib.rs

**Description:** In *preview_shares_in* and *preview_assets_in*, it is possible for the calculation:

```
mul_div(shares_in_scaled, args.max_share_price, SCALED_DECIMALS)?;
```

to overflow the *u64* during the multiplication part, which will cause the shares/assets out to round down to 0.

**Recommendation:** If *assets_in*/*shares_in* is 0, then throw an error.

**Resolution:** Resolved. A zero value check for *assets_in*/*shares_in* was added in *preview_shares_in()* and *preview_assets_in()* as recommended.

#### 5.2.7 All protocol fees can be set to 100% combined

**Severity:** Medium

**Context:** global_pool_fees.rs

**Description:** The function *set_fees* allows the admin to set the *platform_fee*, *referral_fee*, and *swap_fee*. The main check-in place verifies that each fee is less than the *MAX_FEE_BASIS_POINTS*.

```
platform_fee.is_some() && platform_fee.unwrap() > MAX_FEE_BASIS_POINTS
    || referral_fee.is_some() && referral_fee.unwrap() > MAX_FEE_BASIS_POINTS
    || swap_fee.is_some() && swap_fee.unwrap() > MAX_FEE_BASIS_POINTS
```

However, it is possible to configure all the fees, such that the sum of them all will exceed the *MAX_FEE_BASIS_POINTS*.

**Recommendation:** Consider implementing a check that prevents setting the fee values having a sum higher than *MAX_FEE_BASIS_POINTS*.

**Resolution:** Resolved. The Solana program has been updated to check the total fees of *platform_fee*, *referral_fee* and *swap_fee* do not exceed *MAX_FEE_BASIS_POINTS*.

### 5.3 Low

#### 5.3.1 No minimum slippage applied leading to possible loss of funds

**Severity:** Low

**Context:** buy.rs, sell.rs

**Description:** All the swap functions accept a parameter stating the minimum/maximum amount of assets/shares the user is willing to receive/pay in the swap. However, a slippage of zero can be used, allowing possible loss of funds in case a big move in prices occurs.

**Recommendation:** Apply a hard minimum slippage amount in case the user provides a lower value.

**Resolution:** Resolved. A zero slippage check has been added to each buy/sell functions to ensure no slippage values can be set to 0.

#### 5.3.2 Unnecessary token transfers

**Severity:** Low

**Context:** redemption.rs

**Description:** The *close_pool()* function performs the transfer of *pool.total_swap_fees_asset* twice, first to *treasury_asset_token_account* and then to *swap_fee_recipient_asset_token_account*. Similarly, the *pool.total_swap_fees_share* amount is transferred once to *treasury_share_token_account* and then to *swap_fee_recipient_share_token_account*.

**Recommendation:** Directly transfer the amounts to their respective recipients instead of performing each of the transfers twice.

**Resolution:** Resolved. Platform fees and swap fees are now directly transferred to the treasury recipients instead of transferring it to a treasury as a middleman. Do note that if the Fjord team wishes to do so, they can still upgrade the Solana program to utilise the treasury as a proxy recipient if they wish to isolate the treasury recipient distribution from the *close_pool* function logic.