Security Assessment

MoviePass Exchange - MSX Smart Contract

February 2025

Prepared for MoviePass

Table of Contents

Project Summary

Project Scope

Project Name	Repository (link)	Commit Hashes	Platform
MoviePass Exchange	moviepass/msx-smart-contract	Audit start: c0c120d3 Latest fix review: 7961c77c	Sui

Project Overview

This document describes the manual code review of the MoviePass Exchange repository (msx-smart-contract). The work was a 2.5 day-effort undertaken from **10/02/2025** to **13/02/2025.**

The following contract list is included in our scope:

1. `big_vector.move`
2. `exchange.move`
3. `msx.move`
4. `moviepass_tests.move`

The team performed a manual audit of all the smart contracts**.** During the manual audit, the Certora team discovered bugs in the code, as listed on the following page.

Findings Summary

The table below summarizes the findings of the review, including type and severity details.

Severity	Discovered	Confirmed	Fixed
Critical	3	3	3
High	1	1	1
Medium	1	1	0
Low	1	1	0
Total	6	6	4

Severity Matrix

Impact	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Low	Low	Medium
		Low	Medium	High
			Likelihood	

Critical Severity Issues

C-01 DefaultBidPercentage Misconfiguration		
Severity: **High**	Impact: **High**	Likelihood: **High**
Files: exchange.move	Status: Fixed	

- **Impact:** High (affects economic parameters for presale)
- **Probability:** High

Description:
The constant is defined as:

```
const DefaultBidPercentage: u64 = 10_000; // 10% max rate
```

Using `FeeScaling = 1_000_000`, a value of 10,000 represents 1% (10,000/1,000,000 = 0.01), not 10%.

Full Analysis
1. Where is `DefaultBidPercentage` Used?
It appears in `create_curve`:

```
    mut bid_percentage: Option<u64>,
    mut presale_duration: Option<u64>,
...
    let bid_percentage_extracted = if (option::is_some(&bid_percentage)) {
        option::extract(&mut bid_percentage)
    } else {
            admin_settings.default_bid_percentage
        };
...
        let max_b_sold = safe_percentage(b_target_supply,
bid_percentage_extracted);
```

2. How is `bid_percentage_extracted` Applied?

The key operation is:

```
        let max_b_sold = safe_percentage(b_target_supply,
bid_percentage_extracted);
```

Looking at `safe_percentage`:

```
fun safe_percentage(
        number: u64,
        percentage: u64,
): u64 {
        (((number as u128) * (percentage as u128) / FeeScaling) as u64)
}
```

Substituting the values:

- `FeeScaling = 1_000_000` (1e6)
- If `bid_percentage_extracted = 10_000`

Then:

```
safe_percentage(b_target_supply, 10_000)
= (b_target_supply * 10_000) / 1_000_000
= b_target_supply * 0.01
```

This **applies a 1% cap, NOT 10%**.

3. Final Verdict

- `DefaultBidPercentage = 10_000` results in a **1% cap**, **not 10%**.
- There is **no other scaling factor** compensating for this.
- If the **intended cap is 10%**, the value should be `100_000`, not `10_000`.

Recommendation:

- Change the constant to:

```
const DefaultBidPercentage: u64 = 100_000; // 10% max rate
```

- Update documentation and comments to ensure that percentages are expressed in parts-per-million (PPM).

MoviePass's Response: Fixed

C-02 Bid with Zero Input Causing DOS

Severity: **High**	Impact: **High**	Likelihood: **High**
Files: exchange.move	Status: Fixed	

- **Impact:** High (can cause a denial-of-service on the dispersal phase, locking funds for all users)
- **Probability:** High

Description

A bid with a 0-value input can cause the entire dispersal phase to fail if the `split(0)` operation reverts. If a single 0 bid enters the `disperse` function, it may cause a **denial-of-service (DoS)** by failing all subsequent withdrawals.

```
File: exchange.move
0700: let coin_value = coin.value();
0701: let committed_capital_value = curve.presale_total_commits;
0702: let updated_commited_value = coin_value + committed_capital_value;
```

Expected Behavior if a 0-Split Is Disallowed

If the coin module enforces that the split amount must be > 0:

- The function `curve.b_balance.split(0)` would contain an assertion (or equivalent check) and revert when asked to split 0.
- **Impact:**
  - The dispersal function would abort as soon as it encounters a presale commit with 0 value.
  - This would prevent the dispersal transaction from completing and could lock funds in the presale pool.

  - An attacker could deliberately submit a 0-value bid (or bid an amount that rounds to 0 output due to integer division in `get_output_amount`) to trigger this revert.

Worked Example

Assume the following state before dispersal:

- `curve.presale_total_commits` is 100,000 tokens (from all bids combined).
- The `output_amount` computed for the entire presale distribution is 50,000 QUOTE tokens.
- Among the presale commits, there is one commit:
  - `presale_commit.amount = 0` (a zero-value bid).

Now, during dispersal:

1. For the zero bid:
   a. Compute: `amount_b_received = 0 * 50,000 / 100,000 = 0`
2. The code then calls:

```
let coin_b = curve.b_balance.split(0).into_coin(ctx);
```

1. **Case A:** If `split(0)` is allowed (returning a zero-value coin), then this bid is processed as a "no-op" (though it adds an unnecessary entry in the output event list).
2. **Case B:** If `split(0)` is not allowed and the coin module aborts on a 0 split:
   a. The transaction reverts with an error (likely one indicating an invalid split amount).
   b. **Result:** The entire dispersal process aborts—even if all other bids are valid.

Recommendation

- **Enforce a Minimum Bid Value**: Add an assertion ensuring `coin.value() > 0`.
- **Guard in disperse:** Skip processing of any presale commit with a 0 value.

MoviePass's Response: Fixed with a more explicit check but actually there is already a check in get_output_amount that the values are non-zero

Certora's Response: Yes, the check is exactly what's causing the DOS (it isn't in bid, but in the subsequent calls). The fix is still effective as long as min_trade_amount is never zero.

C-03 Bid Stuffing / Micro-bids Griefing Attack		
Severity: **High**	Impact: **High**	Likelihood: **High**
Files: exchange.move	Status: Fixed	

- **Impact:** High (could lead to denial-of-service during token dispersal)
- **Probability:** High (no per-user or per-transaction bid limit)

Description:
In the bid functions (e.g., lines 0687–0721 in the open-loop bid function), there is no minimum bid size or per-user limit.

```
0700:     let coin_value = coin.value();
0701:     let committed_capital_value = curve.presale_total_commits;
0702:     let updated_commited_value = coin_value + committed_capital_value;
0707:     event::emit(BidEvent { ... });
0715:     curve.presale_bal.join(coin.into_balance());
0716:     curve.presale_total_commits = updated_commited_value;
0717:     curve.presale_commits.push_back(PresaleCommit {
0718:         user: ctx.sender(),
0719:         amount: coin_value
0720:     });
```

An attacker can repeatedly submit micro-bids (e.g., 1 unit each) that are valid because the total remains below the cap. This floods the BigVector (`curve.presale_commits`) with many entries. Later, the `disperse` function (lines 0724–0806) must iterate over these entries, significantly increasing gas costs or even causing transactions to fail.

Step-by-Step Attack Execution

1. **Presale Conditions Are Met:**
   a. The curve is in its presale phase (i.e. `clock.timestamp_ms() < curve.presale_end_time`).

   b. The total committed amount so far is well below `curve.presale_max_a`.
2. **Repeated Bid Submission:**
   a. The attacker repeatedly calls the bid function (or its custodial variant) with a coin containing a bid amount of 1 unit.
   b. **Math and Invariants:**
      i. For each bid, the new total becomes: `new_total = old_total + 1`
      ii. As long as the attacker's total (plus other honest bids) remains below `curve.presale_max_a` (which might be, for example, 1,000,000 units), each bid is accepted.
   c. **Access Control:**
      i. The bid function does not restrict how many bids one address may submit. Any address (including an attacker's) is allowed to call the bid function.
      ii. There is no per-transaction or per-user cap.
3. **Effect on the BigVector (Presale Commits):**
   a. Each call to bid appends a new `PresaleCommit` (with `amount = 1` and `user = attacker_address`) to the BigVector.
   b. The BigVector grows by one entry per call.
   c. Since the BigVector is implemented using slices of size 1,000, every 1,000 bids a new slice is allocated.
   d. After, say, 100,000 bids, the BigVector holds 100,000 entries across about 100 slices.
4. **Subsequent Operations and Impact:**
   a. When the presale ends, the `disperse` function (lines 0724–0806) is called. It begins by checking that either the presale end time has passed or that the maximum presale commitment has been reached.
   b. The disperse function then enters a loop to process (pop) bid entries:

```
while (curve.presale_commits.length() > end_target) {
 let presale_commit = curve.presale_commits.pop_back();
 … // process each commit
}
```

- **Pagination:**
  - The function supports an optional `page_size` (defaulting to 1000) so that only the last page of commits is processed per call.

    - However, if the BigVector is huge (e.g. 100,000+ entries), then even with pagination, many separate transactions will be required to process all the bids.
- **Performance Risk:**
    - The cost (gas or computational time) of each disperse call is proportional to the number of entries it processes. A very large BigVector increases the total cost, potentially leading to a denial-of-service (DoS) situation where honest users cannot complete the disperse operation within a single transaction or within acceptable gas limits.

Feasibility Analysis

- **Economic Feasibility:**
    - Since each bid is extremely small (1 unit), the attacker does not need to commit a large total sum. They could, for example, spend 100,000 units total (which might be a small fraction of the total allowed) while generating 100,000 separate entries.
    - If gas costs per bid are low (especially on a network with low congestion), the attacker may be able to submit billions of transactions at a reasonable cost.
- **Logical Feasibility:**
    - There is no built-in mechanism in the bid functions to limit the number of bids from a single address or to aggregate them.
    - The only prevention is the aggregate cap (`curve.presale_max_a`), but by splitting their commitment into many tiny pieces, the attacker can easily avoid reaching that cap while still bloating the commit array.
    - Pagination in disperse helps to limit per-transaction cost but does not prevent the BigVector from growing very large.
- **Access Control:**
    - The presale bid functions are open to any user (or externally controlled address), so the attacker faces no access restrictions.
- **Bypass or Mitigation Considerations:**
    - **No Minimum Bid:** Without a minimum bid amount, the attacker can use arbitrarily small increments.
    - **No Per-User Limit:** There is no check on the number of bids per user.
    - **Pagination Alone Is Insufficient:** While pagination limits the cost per disperse call, it does not prevent the overall BigVector from growing large and causing cumulative delays or high overall gas usage.

Recommendation:

- Set a **minimum bid size** to prevent micro-bids.
- Implement **per-user bid limits** to reduce abuse.
- Aggregate multiple bids from the same user into a single entry.

MoviePass's Response: Fixed. We implemented a minimum bid size. We will not implement per-user bid limits because we are fine taking it with min bid size.
Multiple bids aggregation will be costly from a computation perspective.

High Severity Issues

H-01 Lack of Slippage Protection in Swaps		
Severity: **High**	Impact: **High**	Likelihood: **Medium**
Files: exchange.move	Status: Fixed	

Impact: High (users could receive far less than expected; potential for front-running/sandwich attacks)
Probability: Medium (given volatile markets and AMM dynamics)

Description:
The swap functions (both open loop and custodial) use a simple output calculation via `get_output_amount` (e.g., lines 0829–0833 in `swap_a_to_b`), but they do not include any parameter for a minimum acceptable output.

```
0829:     let output_amount = get_output_amount(
0830:         amount,
0831:         a_balance_value + curve.virtual_a_amount,
0832:         b_balance_value
0833:     );
```

This exposes users to slippage risk if reserve values change between the time of transaction submission and execution.

Recommendation:
- Add an optional parameter (e.g., `min_output: u64`) to the swap functions.
- Before proceeding with the swap, assert that the computed `output_amount` is at least `min_output`.
- Document clearly in the user interface and developer docs how slippage is handled.

MoviePass's Response: Fixed on the min_output and slippage recs.

Certora's Response: In order to fully protect the swap function, a `deadline` parameter should be added in case a user transaction execution gets delayed. This parameter would make the swap transaction revert in case the block timestamp is greater than the supplied value. Without a `deadline` parameter, users' trade may be executed at unexpected times when market conditions are unfavorable. While Sui Move doesn't have a mempool like Ethereum, transactions are still subject to delays based on validator processing.

Medium Severity Issues

M-01 Admin Privilege Abuse (Centralization Risk)		
Severity: **Medium**	Impact: **Very High**	Likelihood: **Low**
Files: exchange.move	Status: Acknowledged	

Description:
Several admin functions (e.g., `admin_create_custodial_pool`, `admin_withdraw_custodial_pool`, `admin_bid_with_custodial_account`) allow an approved manager to operate on any user's custodial pool.

Risk:
- A malicious admin or a compromised manager can withdraw funds from user pools, create pools, or bid on behalf of users without their consent.
- This represents a centralization risk inherent in any system that grants significant power to administrators.

Recommendation:
- Strengthen admin governance, for example by requiring multisig approval or additional confirmation for admin operations.
- Consider logging all admin actions extensively and enabling on-chain or off-chain governance oversight.

MoviePass's Response: Can be upgraded later

Low Severity Issues

L-01 Missing Functionality: Partial Withdrawals		
Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: exchange.move	Status: Acknowledged	

Description:
Both user and admin withdrawal functions (e.g., `withdraw_custodial_pool` and `admin_withdraw_custodial_pool`) currently withdraw the entire balance.

Users who wish to withdraw only a portion of their funds must withdraw everything, then redeposit any funds they wish to keep.
This may lead to increased transaction fees and reduced flexibility.

Relevant lines:

```
1064: let coin_type = type_name::get<T>();
1065: let target = df::borrow_mut<TypeName, Balance<T>>(&mut
custodial_pool.id, coin_type);
1066: let value = target.value();
1067: target.split(value).into_coin(ctx)
```

Recommendation:

- Consider adding an optional parameter to allow partial withdrawal (e.g., `withdraw_amount: u64`).
- Modify the withdrawal functions to support withdrawing only the specified amount, leaving the remainder in the custodial pool.
- If full withdrawals are a deliberate design choice (e.g., for simplicity or to prevent dust accumulation), document this clearly in the user interface and technical documentation.

MoviePass's Response: This is a design choice.

Informational Severity Issues

I-01. DefaultPresaleDuration Misinterpretation

Status: Not Fixed

Description

The constant `DefaultPresaleDuration` is confusing.

Sui timestamps are in milliseconds, so the calculation represents 60 × 60 × 24 × 1000 = 1 day. Either it was intended to be **1,000 days but is actually set to 1 day**, or the units order simply aren't from smallest to biggest.

```
File: exchange.move
29: const DefaultPresaleDuration: u64 = 60 * 60 * 24 * 1000;
```

Recommendation

- Reorder the multipliers for clarity:

```
File: exchange.move
- 29: const DefaultPresaleDuration: u64 = 60 * 60 * 24 * 1000;
+ 29: const DefaultPresaleDuration: u64 = 1000 * 60 * 60 * 24; //@audit 1000
ms * 60 s * 60 min * 24h == 1 day
```

- Clarify the intended presale duration by updating the comments to clearly state that the default presale duration is 1 day.

I-02. Typos and Inconsistent Naming

Status: Fixed

Description:
Several typos exist in comments and function names. Examples include:

- Function name `get_custodial_pool_balanace<T>` (line 1019) should be "get_custodial_pool_balance"
- Field name `b_inital_value` (line 159) should be "b_initial_value."
- Variable `admin_setttings` in tests (line 60 of `moviepass_tests.move`) should be "admin_settings."

MoviePass's Response: Fixed

I-03. Tests Aren't Compiling Due to a Missing Field

Status: Fixed

Description:
In the test file (e.g., `moviepass_tests.move`), a field required by `create_curve` is missing. For example, in the test call, a parameter for the curve name is not provided:

```
66:         let id = exchange::create_curve<MSX, MEME>(
67:             &mut admin_setttings,
68:             treasury_cap,
69:             &coin_metadata,
70:             1000000000,
71:             4000000 * 1_000_000,
72:             option::some(100_000),
73:             option::some(ONE_DAY_MS),
74:             string::utf8(b"ACTOR").to_ascii(),
+ 74:              string::utf8(b"CURVE_NAME").to_ascii(),
75:             &clock,
76:             scenario.ctx()
77:         );
```

The missing parameter causes the tests not to compile, meaning that the current state of the code cannot be fully verified.

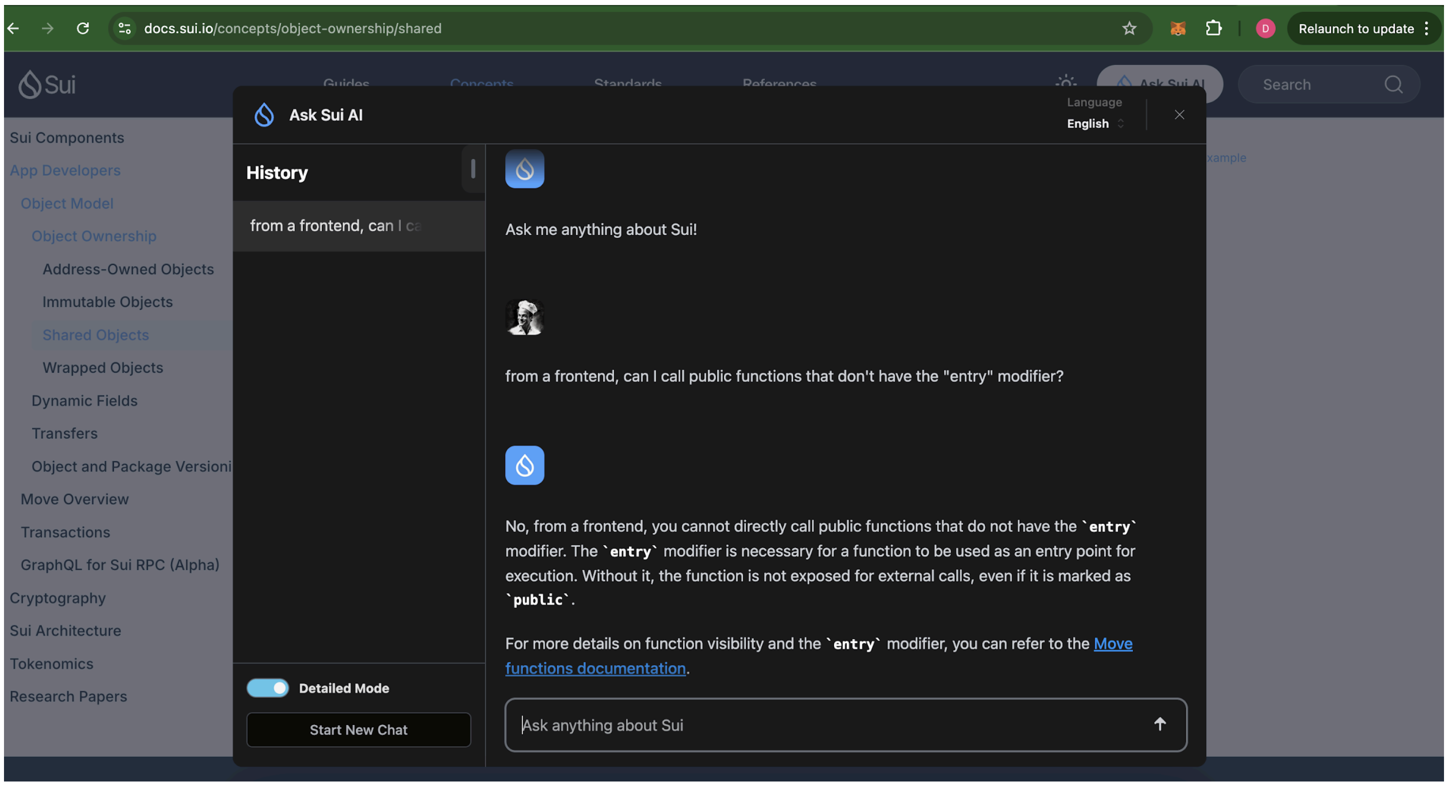
MoviePass's Response: Fixed

Dismissed Concerns

Dismissed-01. No Entrypoints Indicated in the Contracts

Impact: Medium (could affect how the contract is deployed and interacted with)
Probability: Medium
Severity: Medium
Status: Disputed

Description:
None of the functions are marked with an `entry` keyword. In Sui Move, functions that are meant to be directly invoked on-chain typically use `entry`.
This raises the question of whether the contract is intended to be accessed directly or only via transaction scripts or wrappers.

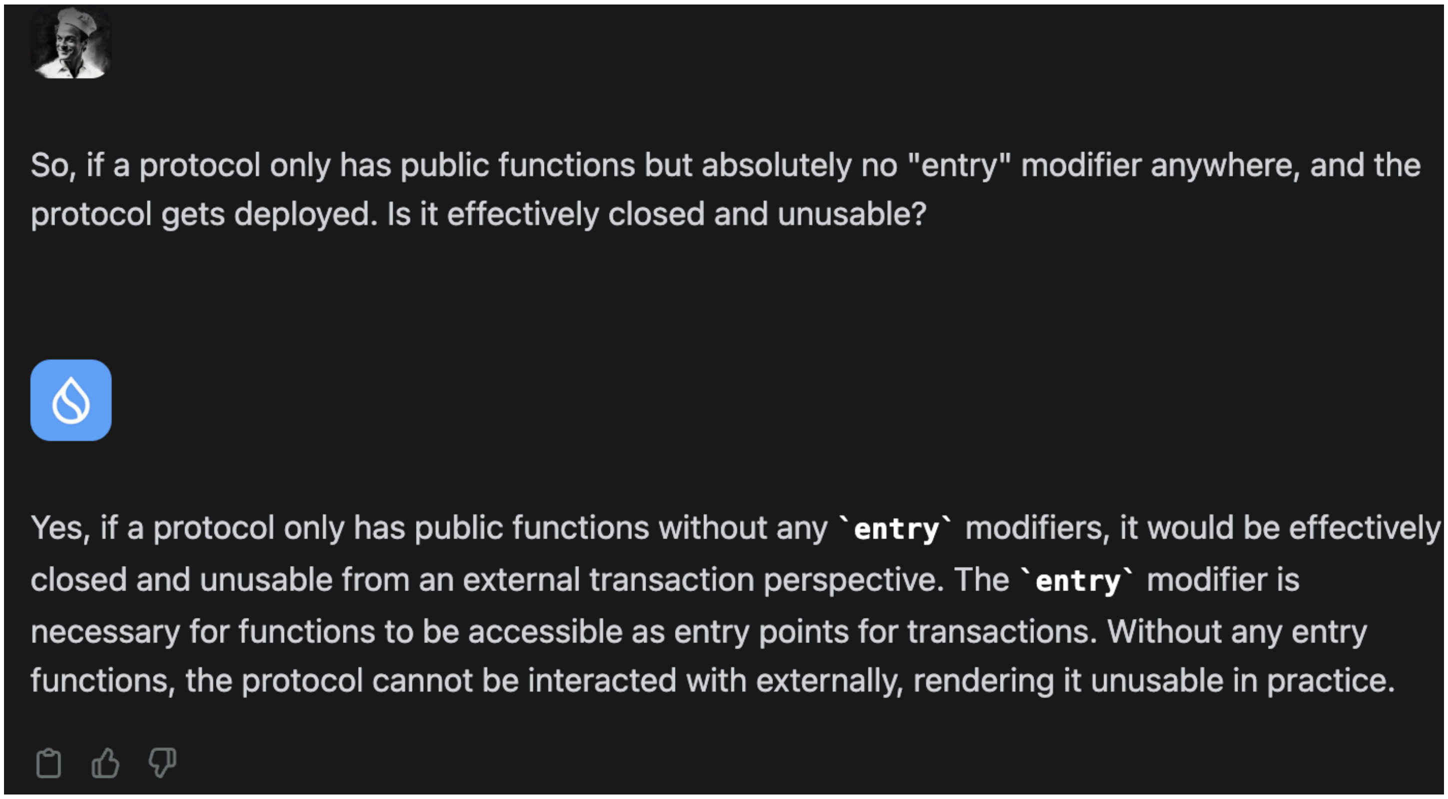


Ask me anything about Sui!

from a frontend, can I call public functions that don't have the "entry" modifier?

No, from a frontend, you cannot directly call public functions that do not have the `` `entry` `` modifier. The `` `entry` `` modifier is necessary for a function to be used as an entry point for execution. Without it, the function is not exposed for external calls, even if it is marked as `` `public` ``.

For more details on function visibility and the `` `entry` `` modifier, you can refer to the Move functions documentation.



So, if a protocol only has public functions but absolutely no "entry" modifier anywhere, and the protocol gets deployed. Is it effectively closed and unusable?

Yes, if a protocol only has public functions without any **`entry`** modifiers, it would be effectively closed and unusable from an external transaction perspective. The **`entry`** modifier is necessary for functions to be accessible as entry points for transactions. Without any entry functions, the protocol cannot be interacted with externally, rendering it unusable in practice.

Recommendation:

- Clarify the intended usage model. If external calls are expected, provide appropriate wrapper functions with the `entry` keyword.
- Update documentation to indicate how the contract is meant to be invoked.

MoviePass's Response: Public has replaced entry functions so we don't need it anymore

Dismissed-02. Lack of Explicit Treasury Management

- **Impact:** High (misallocation of treasury funds can affect token economics)
- **Probability:** Low
- **Severity:** Medium

Description:
In the `init` function (lines 0168–0171), the treasury cap and initial minted tokens are created and transferred without an explicit treasury management structure. For example:

```
0168:     let admin = tx_context::sender(ctx);
0169:     let admin_cap = AdminCap { id: object::new(ctx) };
0170:     transfer::transfer(admin_cap, admin);
```

The treasury (the initial supply of tokens) is blindly transferred to `ctx.sender()`. If the deployer is not the intended recipient (for example, if it should be a governance-controlled multisig), misallocation of funds can occur.

Recommendation:
- Introduce an explicit treasury management mechanism that requires the treasury cap and minted tokens to be transferred to a pre-defined treasury address (or a multisig/governance-controlled account).
- Update the `init` function to validate that the recipient is correct.

MoviePass's Response: Admin cap for this can be sent to multi-sig later if necessary.

Dismissed-03. Prevent Removal of Critical Version

- **Impact:** Medium (may break compatibility with the deployed package version)
- **Probability:** Medium
- **Severity:** Medium

Description:
The functions `remove_version` (lines 0235–0243) do not prevent the removal of the current `PACKAGE_VERSION`:

```
0235: public fun remove_version(
0236:     admin_settings: &mut AdminSettings,
0237:     custodial_pool_idx: &mut CustodialPoolsIndex,
0238:     _cap: &AdminCap,
0239:     version: u64,
0240: ) {
0241:     admin_settings.version_set.remove(&version);
0242:     custodial_pool_idx.version_set.remove(&version);
0243: }
```

Recommendation:

- Add an assertion to prevent removal of the current package version:

```
assert!(version != package_version(), EInvalidPackageVersion);
```

MoviePass's Response: Critical version is the kill switch for the entire protocol.

Dismissed-04. Invariant Maintenance for Manager Set

- **Impact:** Medium (could lead to loss of administrative control)
- **Probability:** Medium
- **Severity:** Medium

Description:
The `remove_manager` function (lines 0200–0206) does not check whether removing a manager would leave the set empty.

If the invariant requires that at least one manager exists at all times, then removal of the final manager could break that invariant.

Relevant lines:

```
0200: public fun remove_manager(
0201:     custodial_pool_idx: &mut CustodialPoolsIndex,
0202:     _cap: &AdminCap,
0203:     manager: address,
0204: ) {
0205:     custodial_pool_idx.managers.remove(&manager);
0206: }
```

Recommendation:
- Before removing a manager, add a check to ensure that the manager set will not become empty, e.g.:

```
assert!(custodial_pool_idx.managers.length() > 1,
ELastManagerRemovalNotAllowed);
```

MoviePass's Response: Invariant for manager set is fine the admin cap will also have access to it.

Disclaimer

Even though we hope this information is helpful, we provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages, or other liability, whether in an action of contract, tort, or otherwise, arising from, out of, or in connection with the results reported here.

About Certora

Certora is a Web3 security company that provides industry-leading formal verification tools and smart contract audits. Certora's flagship security product, Certora Prover, is a unique SaaS product that automatically locates even the most rare & hard-to-find bugs on your smart contracts or mathematically proves their absence. The Certora Prover plugs into your standard deployment pipeline. It is helpful for smart contract developers and security researchers during auditing and bug bounties.

Certora also provides services such as auditing, formal verification projects, and incident response.