Exec Sum

A time-boxed security review of the UnruggableMemecoins protocol was done by **Antoine M.**, with a focus on the security aspects of the application's implementation. Some important vulnerabilities have been found and must be addressed before the deployment of the contracts.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where I try to find as many vulnerabilities as possible. I can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

About me

I'm an independant security researcher for the Starknet ecosystem. Find me on Twitter @Meckerrr and on Github @0xEniotna.

About **UnruggableMemecoins**

UnruggableMemecoins is project lead by the Starkware Exploration team. Unruggable Meme lets creators launch new memecoins with locked liquidity pools, vesting schedules, and other protections baked into the smart contracts.

This prevents many common scam tactics and exit strategies.

Observations

The codebase is clean and well commented. A lot of tests have been written. It interacts a lot with external protocols like Ekubo or Jediswap.

Threat Model

Privileged Roles & Actors

`Factory`: There is an owner but he can do nothing.

`Locker`: The locker has no owner.

`memecoins`: There is an owner. No rights in memecoin contracts. Memecoin owner is used to launch the memecoin and manage LPs.

Security Interview

Severity classification - OWASP

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact - the technical, economic and reputation damage of a successful attack

Likelihood/Difficulty - likelihood or difficulty is a rough measure of how likely or difficult this particular vulnerability is to be uncovered and exploited by an attacker.

Severity - the overall criticality of the risk

Security Assessment Summary

***review commit hash* - 989f075b9133a2ea5a48ecae1a51f1fa0a5345a1**

AND

https://github.com/keep-starknet-strange/unruggable.meme/pull/147

for the ekubo integration. Ekubo's interface has changed during the audit which made the old version obsolete. A quick PR had to be made to make the Ekubo integration work.

Scope

The following smart contracts were in scope of the audit:

- `exchanges/ekubo/ekubo_adapter.cairo` - Done
- `exchanges/ekubo/launcher.cairo` - Done
- `exchanges/jediswap_adapter.cairo` - Done
- `factory/factory.cairo` - Done
- `locker/lock_manager.cairo` - Done
- `locker/lock_position.cairo` - Done
- `token/memecoin.cairo` - Done
- `utils/math.cairo` - Done

The following number of issues were found, categorized by their severity:

- Critical & High: 2 issues
- Medium: 1 issues
- Low: 3 issues
- Optimization: 4 possible
- Informational: 2 possible

Findings Summary

ID	Title	Severity	Status
[C-01]	No buy restriction on Jediswap	Critical	
[C-02]	Malicious counterparty token	Critical	
[M-01]	Liquidity not locked on ekubo	Medium	
[L-01]	Unmodified variable	Low	
[L-02]	Wrong CEI pattern	Low	
[L-03]	Multicall function blocks complex routes	Low	
[L-04]	Deprecated methods	Low	
[O-01]	Useless check on sender	Optimization	
[O-02]	Useless Ownable implementation in memecoin	Optimization	
[O-03]	Useless Ownable implementation in factory	Optimization	
[O-04]	Useless is_launched check in factory	Optimization	
[I-01]	Variable not written after usage	Informational	
[I-02]	No withdrawal Ekubo	Informational	

Detailed Findings

[C-01] No buy restriction on Jediswap

Severity - Critical

Impact: High

Likelihood: High

Description

Contract: `memecoin.cairo` Function: `apply_transfer_restrictions`

According to the comment, this function Applies the relevant transfer restrictions, if the timing for restrictions has not elapsed yet.

- Before launch, the number of holders and their allocation does not exceed the maximum allowed.
- After launch, the transfer amount does not exceed a certain percentage of the total supply.

and the recipient has not already received tokens in the current transaction.

The initial goal was to add a cap in the token amount a user can buy. **Considering only Jediswap for the issue** The transfer/transferFrom functions can be called in the following scenarios: swap tokens, add/withdraw liquidity, basic token transfer. In the context of adding or withdrawing liquidity, there is no issue. The pair is whitelisted correctly. For a basic transfer, the restrictions are applied. For a swap, there are only two directions `memecoins` to `ETH` for a sell, `ETH` to `memecoins` for a buy. For a sell, there is no issue. The buy process is the following: user interacts with jediswap router, router does a `transferFrom(user, pair, amount of ETH)`. Then the pair has to transfer an equivalent amount of tokens to the user `IERC20.transfer(contract_address=memecoin, recipient=to, amount=amount1Out);`.

In this block from `apply_transfer_restrictions`, if `get_caller_address` is the pair then their is no restrictions.

```
LiquidityType::ERC20(pair) => {
    if (get_caller_address() == pair || recipient == pair) {
        return;
    }
},
```

It is precisely the case for a buy. Hence, there is in fact no buy limit !

A POC to prove this could simply consist in doing this in a test like `tests/fork_tests/test_jediswap.cairo/test_jediswap_integration`:

```
    let amount_in = 2 * pow_256(10, 17); // @audit Bigger amount
    'amount_eth'.print();
    amount_in.print();
    'before_owner_bal'.print();
    memecoin.balanceOf(owner).print();
    start_prank(CheatTarget::One(router.contract_address), owner);
    let first_swap = router
        .swap_exact_tokens_for_tokens(
            amountIn: amount_in,
            amountOutMin: 0,
            path: array![quote_address, memecoin_address],
            to: owner,
            deadline: starknet::get_block_timestamp()
        );
    'max_tokens'.print();
    let max_tokens: u256 = PercentageMath::percent_mul(
        DEFAULT_INITIAL_SUPPLY(), MAX_PERCENTAGE_BUY_LAUNCH.into()
    );
    max_tokens.print();
    'swapped_owner_bal'.print();
    memecoin.balanceOf(owner).print();
```

[C-02] Malicious counterparty token

Severity - Optimization

Impact: High

Likelihood: High

Description

A pair can be created with any token. Hence a malicious token could be used for the counterparty. If we cant manipulate the Unruggable memecoins, we can manipulate the counterparty ! I would prefer to have a whitelist (i.e starkgate tokens, ekubo tokens or jediswap tokens). Even if it's not a direct vulnerability of Unruggable coinbase, it would have a severe impact on the project credibility

[M-01] Liquidity not locked on ekubo

Severity - Medium

Impact: High

Likelihood: Low

Description

Contract: `ekubo/launcher.cairo` (or `factory.cairo` or `ekubo_adapter.cairo`, depending on where the solution should be implemented) Function: `launch_token` or `locked`

When we create a pool on Ekubo, the liquidity is not locked afterwards. It is minted and the `launchpad` remains the owner of the NFT. To withdraw a position on Ekubo, the owner of the NFT or an approved account can call `positions.withdraw()`. It is the exact same process than the one implemented to withdraw the fees

I couldn't write a succesful attack POC since the `launcher` is the NFT owner, so I tagged this one as a Medium severity (**it would be critical otherwise**).

It would be reassuring to have the liquidity locked in the locker. The owner of a position could *maybe* find a way of withdrawing the liquidity.

[L-01] Unmodified variable

Severity - Low

Description

Contract: `lock_manager.cairo` Function: `increase_lock_amount`

```
let mut token_lock = self.locks.read(lock_address);
self.locks.write(lock_address, token_lock);
```

We read a variable before writing it back without doing anything. Either an operation has been forgotten, either it is useless to perform the `write()`.

[L-02] Wrong CEI pattern

Severity - Low

Description

Contract: `lock_manager.cairo` Function: `partial_withdraw`

There is a error in the implementation of the Check-effect-interactions pattern in this function. The if block `if actual_balance == amount` should be before the `transferFrom`.

[L-03] Multicall function blocks complex routes

Severity - Low

Description

Contract: `memecoin.cairo` Function: `ensure_not_multicall`

The current multicall blocker will panic iif the transfer is perform using a complex route like Avnu could do. A simple POC for this would be to write exactly the same test as one would do to test the swap on ekubo and perform 2 swaps with `amount_in / 2` as input. Avnu sometimes splits the swap in 2.

Note that regarding Jediswap, the `ensure_not_multicall` function will simply never be called when buying tokens, see [C-01].

[L-04] Deprecated methods

Severity - Low

Description

Contract: `launcher.cairo` Function: `withdraw_fees`

The current implementation of `withdraw_fees` uses `withdraw` in the callback. The new version of Ekubo's `positions` contract has a special function for this: `collect_fees`.

We should now use `collect_fees` to collect the fees and `withdraw_v2` to withdraw liquidity from a position.

[O-01] Useless check on sender

Severity - Optimization

Description

Contract: `memecoin.cairo` Function: `enforce_prelaunch_holders_limit`

```
if sender.is_non_zero() && self.balanceOf(sender) == amount {
```

sender will never be 0, we can save gas removing this.

We'll probably not save gas though.

[O-02] Useless Ownable implementation in memecoin

Severity - Optimization

Description

Contract: `memecoin.cairo`

Ownable is implemented but it is actually never used in `memecoin`. The only use of `owner` is in factory to check who is calling `launch_on_ekubo` or `launch_on_jediswap`. Seems like we can save gas by replacing this by a single `owner` variable in the storage isntead of using the entire ownable component.

[O-03] Useless Ownable implementation in factory

Severity - Optimization

Description

Contract: `factory.cairo`

Ownable is implemented but it is actually never used in `factory.cairo`.

We can save some gas there.

[O-04] Useless is_launched check in factory

Severity - Optimization

Description

Contract: `factory.cairo` Functions: `launch_on_jediswap` and `launch_on_ekubo`.

These 2 functions perform this check: `assert(!memecoin.is_launched(), errors::ALREADY_LAUNCHED);`. It is already performed in `memecoin.set_launched()` which is called in both functions.

We can save a bit of gas here.

[I-01] Variable not written after usage

Severity - Informational

Description

Contract: `launcher.cairo` Functions: `launch_token`. This pattern also happens in the `lock_manager`.

Not an issue since we don't need to write when we use alexandria's List but here, the variable owner_positions is read but not written after the modification. I think we should advocate for performing a write every time, to avoid confusing scenarios where the updated variable is sometimes written afterwards and sometimes not.

```
let mut owner_positions = self.owner_to_positions.read(params.owner);
owner_positions.append(id);
```

[I-02] No withdrawal Ekubo

Severity - Informational

Description

Related to **C-02**. No function exists to allow the owner of an Ekubo position to withdraw the liquidity while it is implemented for JediSwap.