Unruggable Memecoin Security Review

Conducted by: Erim V.

Jan. 2024

Commit Hash: v0.1.0-alpha.2

Disclaimer

This smart contract audit report is provided for informational purposes only and does not constitute investment advice, financial guidance, or any other type of endorsement. The audit aims to identify potential vulnerabilities and issues within the smart contract code as of the date of the audit. However, it is important to understand that this audit does not guarantee the security or functionality of the smart contract and should not be seen as a comprehensive assurance of the code's integrity. Erim V. accepts no liability for any losses or damages arising from the use of this report or the smart contracts in question. Users of the smart contract are solely responsible for conducting their own due diligence and assuming all risks associated with its use.

Introduction

This document presents a detailed analysis and review of the smart contracts in question, aimed at identifying any potential security vulnerabilities, operational inefficiencies, and compliance issues. This audit was conducted just by Erim V. (0xErim).

Smart contracts are immutable once deployed on the blockchain, making it crucial to ensure their robustness and security prior to deployment. Our audit process involved a thorough examination of the contract's codebase, design patterns, and its interaction with both the blockchain and external entities. We focused on verifying the contract's adherence to specified requirements, its resilience against common attack vectors, and its overall performance efficiency.

Protocol Review

Unruggable Memecoin is a protocol developed by the developer community of Starknet. The protocol aims to let people launch their own meme coins without any development efficiency and securely for both users and themselves. Protocol has a factory contract that deploys and launches meme coins, which is the main entry point of the whole system. The protocol also has its own liquidity launching mechanism that integrates the Ekubo Protocol and Jediswap. Memecoin developers can initiate their liquidity without permission under factory contracts. Factory contract is ownable, but there is no use case, and there aren't any protocol-controlled methods in the whole protocol.

Risk Classification

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

Likelihood

High: The attack vector is possible with reasonable assumptions and mostly simple to achieve on-chain conditions, cost of the attack is relatively low compared to the amount of loss done.
Medium: The attack vector needs specific conditions, but still relatively likely.
Low: There are many dependant assumptions that require significant conditions.

Impact

High: Causes significant loss of funds, losing ownership of contract, and changes user data without permission.
Medium: A small amount of funds can be lost, or the core functionality of the protocol is affected.
Low: Leads any kind of unexpected behavior with some of the protocol functionalities that are not critical.

Code improvements like gas optimizations and unnecessary codes are reported as **Best Practices** or **Informational**.

Executive Summary

This report will present findings about the protocol **Unruggable Memecoin**. There will be a detailed representation of 16 findings in total. 1 High, 1 Medium, 3 Low, 4 Informational, and 7 Best Practices findings as grouped according to risk classification. In the final version of the report, there is also a status update for each fix in the below of each finding.

Findings

ID	Title	Severity	Status
H-01	Ekubo adapter can used to receive non-launched tokens	**High**	Unresolved
M-01	Transfer restrictions are not working as expected	**Medium**	Unresolved
L-01	Factory can be used to create liquidity for third-party tokens	**Low**	Unresolved
L-02	Multicall protection will block aggregator calls	**Low**	Unresolved
L-03	Multicall protection can be bypassed	**Low**	Unresolved
I-01	Launch holder count can be higher than actual unique holders	**Informational**	Unresolved
I-02	Ownable component features never used	**Informational**	Unresolved
I-03	Ekubo router has to be immutable	**Informational**	Unresolved
I-04	*max_percentage_buy_launch* can be higher than 10000	**Informational**	Unresolved
BP-01	Functions *self* parameter can be snap instead of reference	**Best Practices**	Unresolved
BP-02	Unused system call	**Best Practices**	Unresolved
BP-03	Launch time and delay can be used as one variable	**Best Practices**	Unresolved
BP-04	Unnecessary storage write	**Best Practices**	Unresolved
BP-05	Same assertion statement used twice	**Best Practices**	Unresolved
BP-06	Unnecessary call	**Best Practices**	Unresolved
BP-07	Unused variables	**Best Practices**	Unresolved

H-01 Ekubo adapter can used to receive non-launched tokens

File(s): factory.cairo, ekubo_adapter.cairo

Description: Malicious actors can create memecoin and use non-launched memecoin as a quote token to launch on Ekubo protocol. However, the factory sends memecoin to Ekubo protocol to create liquidity and then sends the rest quote token to the caller via *clear(...)* method inside the *create_and_add_liquidity(...)* method.

```
fn clear(token: ContractAddress) {
    let caller = starknet::get_caller_address();
    let this = starknet::get_contract_address();
    let token = ERC20ABIDispatcher { contract_address: token, };
    token.transfer(caller, token.balanceOf(this));
}
```

Recommendation: Consider checking quote token is also memecoin.

M-01 Transfer restrictions are not working as expected

File(s): memecoin.cairo

Description: *apply_transfer_restrictions(...)* method early returns if the caller or receiver is a Jediswap pair. However, if the Jediswap pair is the caller, that means tokens are sent from Jediswap, which also means it is a buy transaction. Also, if the pair is the receiver, then it is a sell transaction. In expected behavior, restrictions have to be applied in these conditions. However, if these conditions are met, it early returns.

```
match liquidity_type {
    LiquidityType::ERC20(pair) => {
        if (get_caller_address() == pair || recipient == pair) {
// @audit : Early returns if it is buy or sell tx
            return;
        }
    },
    LiquidityType::NFT(_) => {}
}
```

Recommendation: Consider applying restrictions only when conditions are met.

L-01 Factory can be used to create liquidity for third-party tokens

File(s): factory.cairo

Description: Factories *launch_on_jediswap*, and *launch_on_ekubo* are not checking for the *memecoin_address* deployed by the factory. Malicious actors can use this function to create liquidity for their own tokens, which causes safe event emitting from factory contract. There won't be an on-chain effect, but this will be read by off-chain listeners, which can cause fake and unsafe tokens seems to be launched from this protocol.

```
self
    .emit(
        MemecoinLaunched {
            memecoin_address, quote_token: quote_address, exchange_name:
'Jediswap'
        }
    );
```

Recommendation: Consider checking whether *memecoin_address* is deployed by the factory or not. *is_memecoin* function can be used for that.

L-02 Multicall protection will block aggregator calls

File(s): memecoin.cairo

Description: Multicall protection in the method *ensure_not_multicall* is implemented to block users from getting multiple buys in the same transactions. However, aggregators like AVNU and Fibrous are first taking tokens into their own contracts and then sending them back to users. In that case, the following statement reverts the transaction.

```
assert(self.tx_hash_tracker.read(tx_origin) != tx_hash, 'Multi calls not
allowed');
```

Recommendation: Consider whitelisting aggregator addresses nor removing this check.

L-03 Multicall protection can be bypassed

File(s): memecoin.cairo

Description: The purpose of multicall protection is to limit users buying in the same transaction. However, users can still send multiple different transactions to buy tokens in the same block but not in the same transaction. In fact, it doesn't limit that.

```
#[inline(always)]
fn ensure_not_multicall(ref self: ContractState) {
    let tx_info = get_tx_info().unbox();
    let tx_hash = tx_info.transaction_hash;
    let tx_origin = tx_info.account_contract_address;
    assert(self.tx_hash_tracker.read(tx_origin) != tx_hash, 'Multi calls
not allowed');
    self.tx_hash_tracker.write(tx_origin, tx_hash);
}
```

Recommendation: Consider limiting buy amount per account in the same block.

I-01 Launch holder count can be higher than actual unique holders

File(s): memecoin.cairo

Description: While initializing memecoin, *check_and_allocate_team_supply(...)* method allocates team tokens and mintings. However, this function does not check *initial_holders* parameter has the same address multiple times. This causes the wrong value for *pre_launch_holders_count*.

```
self.pre_launch_holders_count.write(initial_holders.len().try_into().unwra
p()); //@audit initial_holders can contain the same addresses
```

Recommendation: Consider increasing this value only if the address is unique.

I-02 Ownable component features never used

File(s): factory.cairo

Description: Ownable component implemented but never used by factory contract.

Recommendation: Consider removing the initialization of unused component.

I-03 Ekubo router has to be immutable

File(s): factory.cairo

Description: Ekubo router address has to be dynamic. As noticed, Ekubo updating their router address frequently. In case the router address changes, then this address also has to be updated in the factory contract.

Recommendation: Consider reading router address dynamically from Ekubo.

I-04 *max_percentage_buy_launch* can be higher than 10000

File(s): memecoin.cairo

Description: *max_percentage_buy_launch* value is set inside *set_launched(...)* function. However, the upper bound of the value wasn't checked.

```
assert(
    max_percentage_buy_launch >= MIN_MAX_PERCENTAGE_BUY_LAUNCH, //@audit
Can be higher than 10000
    errors::MAX_PERCENTAGE_BUY_LAUNCH_TOO_LOW
);
```

Recommendation: Consider checking value before updating storage.

BP-01 Functions self parameter can snap be used instead of reference

File(s): launcher.cairo

Description: The following functions can be used as a view method. To achieve this, use self parameter as Snap, not reference.

```
fn launched_tokens(ref self: ContractState, owner: ContractAddress) ->
Span<u64> {  // @audit Read only method. self to Snap
    // ...
}

fn liquidity_position_details(ref self: ContractState, id: u64) -> EkuboLP
{ // @audit Read only method. self to Snap
    // ...
}
```

Recommendation: Consider modifying *ref self: ContractState* to *self: @ContractState*

BP-02 Unused system call

File(s): ekubo_adapter.cairo

Description: get_caller_address(...) system call called but never used.

```
let caller_address = starknet::get_caller_address(); // Line: 60
```

Recommendation: Consider removing this line.

BP-03 Launch time and delay can be used as one variable

File(s): memecoin.cairo

Description: *launch_time* and *transfer_restriction_delay* can be stored in one slot as a sum. Before launch, both values are zero.

Recommendation: Consider summarizing these values into one value.

BP-04 Unnecessary storage write

File(s): lock_manager.cairo

Description: Storage variable written back to storage without changes.

```
fn increase_lock_amount(
    ref self: ContractState, lock_address: ContractAddress,
amount_to_increase: u256
) {
    // ...
    let mut token_lock = self.locks.read(lock_address);
    self.locks.write(lock_address, token_lock); //@audit Unnecessary write
    // ...
}
```

Recommendation: Consider removing the storage write call.

BP-05 Same assertion statement used twice

File(s): factory.cairo

Description: The following assert statement was used twice in function *launch_on_jediswap(...).*

```
assert(!memecoin.is_launched(), errors::ALREADY_LAUNCHED);
```

Recommendation: Consider removing one of these assertion statements.

BP-06 Unnecessary call

File(s): jediswap_adapter.cairo

Description: Quote tokens *balanceOf(...)* method called but returned value never used in function *create_and_add_liquidity(...)*.

```
let quote_balance = quote_token.balanceOf(this); // @audit : Unnecessary
```

Recommendation: Consider removing this line.

BP-07 Unused variables

File(s): factory.cairo

Description: Quote token dispatchers are not used in both *launch_on_ekubo(...)* and *launch_on_jediswap(...)*

```
let quote_token = ERC20ABIDispatcher { contract_address: quote_address };
```

Recommendation: Consider removing these lines.