Security Audit Report

memes.fun

v1.0

September 24, 2025

Table of Contents

Table of Contents **2**
License **4**
Disclaimer **5**
Introduction **6**
Purpose of This Report 6
Codebase Submitted for the Audit 6
Methodology 7
Functionality Overview 7
How to Read This Report **8**
Code Quality Criteria **9**
Summary of Findings **10**
Detailed Findings **12**
1. Missing enforcement for half_tokens_to_migrate before liquidity provision 12
2. Migrations will fail due to undispatched messages 12
3. Static price calculation allows price manipulation and unfair trading 13
4. Usage of floating numbers is disallowed by the CosmWasm VM 13
5. Iteration over token holders could cause a denial of service 14
6. Migration pool creation fails if the pool already exists 14
7. Reserve update does not include fee deduction 15
8. Updating settings parameters after deployment may cause miscalculations 15
9. Unnecessary division when calculating fee amounts 16
10. Fee calculation may be bypassed with small payments 16
11. Incorrect slippage tolerance logic validation 17
12. Last buyers will be forced to use poor exchange rates 17
13. Incorrect validation of supply restriction limits 18
14. Updating settings on deployed contracts may fail due to gas limits 18
15. Potential leftover funds due to imprecise liquidity splits 18
16. Instantiate lacking proper validations 19
17. Ignored error during POOL.update 19
18. Query functions do not paginate responses 20
19. Potential query failures due to large entries in the TRANSACTION_HISTORY 21
20. Parsing failures in calculate_price will default to the base price 21
21. The is_apply_on_new_contracts flag should be set to true if fee values change 22
22. Conversion for the target market cap can be simplified 22
23. Funds validation can be improved 22
24. Functions throughout the codebase return minimal or no response attributes 23
25. Incorrect error field returned in extract_token_response_data 24
26. Usage of magic numbers 24
Appendix **24**

1. Test case for “Incorrect slippage tolerance logic validation” 25

License

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

THIS AUDIT REPORT WAS PREPARED EXCLUSIVELY FOR AND IN THE INTEREST OF THE CLIENT AND SHALL NOT CONSTRUE ANY LEGAL RELATIONSHIP TOWARDS THIRD PARTIES. IN PARTICULAR, THE AUTHOR AND HIS EMPLOYER UNDERTAKE NO LIABILITY OR RESPONSIBILITY TOWARDS THIRD PARTIES AND PROVIDE NO WARRANTIES REGARDING THE FACTUAL ACCURACY OR COMPLETENESS OF THE AUDIT REPORT.

FOR THE AVOIDANCE OF DOUBT, NOTHING CONTAINED IN THIS AUDIT REPORT SHALL BE CONSTRUED TO IMPOSE ADDITIONAL OBLIGATIONS ON COMPANY, INCLUDING WITHOUT LIMITATION WARRANTIES OR LIABILITIES.

This audit has been performed by

Oak Security GmbH

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security GmbH has been engaged by memes.fun to perform a security audit of the memes.fun smart contracts.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following target:

Repository	https://github.com/mdfadmin/mdf_cw25
Commit	`822379bfc696d9feb364cad4498f530e3d9881db`
Scope	Only the `contracts/deployer` and `contracts/token` directories were in the scope of the audit.
Fixes verified at commit	`af334c621512f1ea03085405246e4a53e9718289` Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

memes.fun is a platform where users can create and launch tokens without writing any code. Traders can buy and sell these tokens using a bonding curve mechanism. Once a token reaches its target market cap, the contract's liquidity is deployed to a decentralized AMM exchange.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending**, **Acknowledged**, **Partially Resolved**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	Low-Medium	-
Level of documentation	Low	-
Test coverage	Medium-High	`cargo tarpaulin` reports a code coverage of `79.69%`.

Summary of Findings

No	Description	Severity	Status
1	Missing enforcement for `half_tokens_to_migrate` before liquidity provision	Critical	Resolved
2	Migrations will fail due to undispatched messages	Critical	Resolved
3	Static price calculation allows price manipulation and unfair trading	Critical	Resolved
4	Usage of floating numbers is disallowed by the CosmWasm VM	Critical	Resolved
5	Iteration over token holders could cause a denial of service	Critical	Resolved
6	Migration pool creation fails if the pool already exists	Major	Resolved
7	Reserve update does not include fee deduction	Major	Resolved
8	Updating settings parameters after deployment may cause miscalculations	Major	Resolved
9	Unnecessary division when calculating fee amounts	Major	Resolved
10	Fee calculation may be bypassed with small payments	Major	Resolved
11	Incorrect slippage tolerance logic validation	Major	Resolved
12	Last buyers will be forced to use poor exchange rates	Major	Resolved
13	Incorrect validation of supply restriction limits	Major	Resolved
14	Updating settings on deployed contracts may fail due to gas limits	Minor	Resolved
15	Potential leftover funds due to imprecise liquidity splits	Minor	Resolved
16	Instantiate lacking proper validations	Minor	Resolved
17	Ignored error during `POOL.update`	Minor	Resolved

18	Query functions do not paginate responses	Minor	Resolved
19	Potential query failures due to large entries in the `TRANSACTION_HISTORY`	Minor	Resolved
20	Parsing failures in `calculate_price` will default to the base price	Minor	Resolved
21	The `is_apply_on_new_contracts` flag should be set to true if fee values change	Minor	Resolved
22	Conversion for the target market cap can be simplified	Informational	Resolved
23	Funds validation can be improved	Informational	Resolved
24	Functions throughout the codebase return minimal or no response attributes	Informational	Acknowledged
25	Incorrect error field returned in `extract_token_response_data`	Informational	Resolved
26	Usage of magic numbers	Informational	Resolved

Detailed Findings

1. Missing enforcement for `half_tokens_to_migrate` before liquidity provision

Severity: Critical

In `contracts/token/src/contract/reply.rs:85-101`, the `save_migration` function sends `half_tokens_to_migrate` as part of the liquidity provision without verifying that this amount is present in the contract's balance. The current implementation does not enforce any logic that ensures the tokens exist beforehand.

Consequently, the migration process will always fail due to insufficient balance errors, causing a denial of service.

Recommendation

We recommend enforcing the `half_tokens_to_migrate` requirement by validating against `settings.allocated_supply_for_migration` and minting the base tokens to the contract before attempting liquidity provision.

Status: Resolved

2. Migrations will fail due to undispatched messages

Severity: Critical

In `contracts/token/src/contract/exec.rs:333-335`, the `migrate_token` function is called if the current market cap equals or exceeds `settings.target_market_cap`.

The issue is that the return value from `migrate_token` is not handled. Since the `migrate_token` function will dispatch messages (see `contracts/token/src/contract/exec.rs:502`), they will never be executed. This means that any intended migration actions will be ineffective, causing a silent failure of the migration process.

Recommendation

We recommend appending the `Response` from `migrate_token` so that messages are dispatched.

Status: Resolved

3. Static price calculation allows price manipulation and unfair trading

Severity: Critical

In the `calculate_price` function in `contracts/token/src/contract/utils.rs:148-172`, the price calculation is based solely on total circulating supply without considering the purchase amount. This creates a fundamentally flawed bonding curve where price remains static for any given supply level, regardless of attempted trade size, impacting the buy and sell functionalities.

The fundamental issue is that the price calculation is static with the amount of circulating supply before the transaction, so on the buy side, this gives large buyers slippage-free swaps, and likewise on the sell side.

The current price calculation mechanism enables a sophisticated attack where malicious actors can accumulate tokens across multiple addresses and then dump them at a single price point, bypassing the intended bonding curve mechanics. This works because the price is calculated using circulating supply before the trade, not considering trade size, allowing large sells to execute at one price point instead of sliding down the curve.

In a proper AMM, selling a large percentage of supply would incur massive slippage (e.g., 50%+ price impact depending on pool size), but the current implementation provides no slippage protection, enabling attackers to drain the reserve at artificially inflated prices and extract maximum value while legitimate users lose value to manipulative trading strategies.

This issue is also present in the static market cap price calculation found in:

- `contracts/token/src/contract/exec.rs:325-330`
- `contracts/token/src/contract/exec.rs:386`

Recommendation

We recommend implementing a dynamic pricing model. One example reference could be the meteora bonding curve code. The simplest solution is to include the trade amount as a ratio of the reserve in the price calculation, but this would not be a true dynamic bonding curve.

Status: Resolved

4. Usage of floating numbers is disallowed by the CosmWasm VM

Severity: Critical

In `contracts/token/src/utils.rs:148-164`, the `calculate_price` function converts `Uint128` and `Decimal` values to `f64` for computation. This is problematic because CosmWasm's VM forbids floating-point operations due to deterministic execution enforcement.

Consequently, the contract will be undeployable on-chain, as Wasm bytecode containing floating-point operations will be rejected by CosmWasm.

Recommendation

We recommend replacing all `f64` usage with fixed-point decimal types supported by CosmWasm, such as `Decimal` or `Decimal256`.

Status: Resolved

5. Iteration over token holders could cause a denial of service

Severity: Critical

In the token contract, the code iterates over all tokenholders (from `TOKEN_HOLDERS` state) using linear search operations `.iter` when processing buy/sell operations. This is problematic because it creates a critical scalability issue where the gas cost and execution time grow linearly with the number of tokenholders, potentially causing transactions to fail due to out-of-gas errors.

In practice, this condition can occur naturally or be artificially created if a single entity were to purchase small amounts on many addresses. This iteration occurs twice during the buying process, which further increases the likelihood of an out-of-gas denial of service being reached.

- `contracts/token/src/contract/exec.rs:231`
- `contracts/token/src/utils.rs:31`

Additionally, this impacts the reliability of the `get_user_balance` query in `contracts/token/src/contract/query.rs:205`.

Recommendation

We recommend converting `TOKEN_HOLDERS` to a `Map` for `O(1)` lookups rather than `O(n)` iterations. The map entry will use the user address as the key, with its value as `TokenHolders`.

Status: Resolved

6. Migration pool creation fails if the pool already exists

Severity: Major

In `contracts/token/src/contract/exec.rs:481-501`, the `migrate_token` function intends to create a new DEX pair by sending a `create_pair` message to

`settings.dex_contract`. The `settings.dex_contract` field is an Oroswap factory DEX contract that oversees all pair contracts instantiated.

The issue is that the factory contract prohibits the creation of duplicate pools. If the pair has already been created, the factory contract will return an error. This is problematic because an attacker can cause a denial of service by purchasing the base tokens and pairing with `uzig` to create the pool beforehand, which would force the migration process to fail.

Recommendation

We recommend ensuring that unauthorized users cannot create the pair contract, and only the token contract can create it.

Status: Resolved

7. Reserve update does not include fee deduction

Severity: Major

In `contracts/token/src/contract/exec.rs:401`, the `sell_token` function deducts `zigs_in_reserve` by `remaining_zig_to_return` instead of `total_zig_to_return`. This means the deducted amount excludes the fee, even though the fee is also taken from the reserve and sent to the treasury in line `401`.

Consequently, the `zigs_in_reserve` state will become inaccurate as the actual contract balance is different from it. This may cause transactions to fail due to insufficient balance errors.

Recommendation

We recommend deducting `zigs_in_reserve` by `total_zig_to_return`.

Status: Resolved

8. Updating settings parameters after deployment may cause miscalculations

Severity: Major

In `contracts/token/src/contract/init.rs:81-93`, the `update_settings` function allows updates to key token configuration values after the token has been deployed, such as `max_supply`, `allocated_supply_for_sale`, and `base_price`. This is problematic because updating these fields may be ineffective and could cause miscalculations.

For example, `settings.max_supply` is used when creating the denom to define the minting cap. Updating it later will be ineffective because the cap is already configured in `contracts/token/src/contract/exec.rs:65`, causing an inconsistency.

The `settings.base_price` and `settings.curve_steepness` fields are used in `buy_token` and `sell_token` functions to compute the price (see `contracts/token/src/contract/exec.rs:182-190`). Updating them will cause users who previously purchased at a different configuration to face inconsistent pricing.

Recommendation

We recommend disallowing updates to critical parameters once the token is deployed.

Status: Resolved

9. Unnecessary division when calculating fee amounts

Severity: Major

In `contracts/token/src/contract/exec.rs:213` and `393`, the current calculation divides `(payment * settings.trading_fee_percentage)` by `Decimal::from_ratio(100u128, 1u128)`. This is problematic because the `settings.trading_fee_percentage` field is already stored as a `Decimal` in percentage form (e.g., `Decimal::percent(2)` for 2% in `contracts/deployer/src/contract/test.rs:32`).

Consequently, a double division will occur, causing the fee amount to be less than intended.

Recommendation

We recommend removing the unnecessary divisions.

Status: Resolved

10. Fee calculation may be bypassed with small payments

Severity: Major

In `contracts/token/src/contract/exec.rs:211-213` and `393-397`, the trading fee is computed as payment multiplied by `settings.trading_fee_percentage`. Due to integer truncation, the result will be floored when converted to integer amounts. If the payment amount is small, the calculated fee may round to zero, effectively bypassing the fee mechanism.

This allows users to split trades into small payments to avoid paying fees, leading to revenue loss for the protocol.

Recommendation

We recommend using `Decimal::ceil()` when computing fees to ensure that fees are always incurred.

Status: Resolved

11. Incorrect slippage tolerance logic validation

Severity: Major

In `contracts/token/src/contract/exec.rs:255-259`, the slippage check calculates `minimum_token_to_mint` as `token_to_mint * slippage_tolerance`, effectively multiplying `token_to_mint` by the slippage tolerance.

However, slippage tolerance is typically a divisor, not a multiplier. For example, a 5% slippage tolerance indicates the buyer is willing to accept up to 1.05× the price (or 0.95× the tokens), and calculations should reflect division logic.

Consequently, the slippage tolerance enforcement will not work as expected, potentially causing a loss of funds for buyers.

Please refer to the `test_slippage_not_working` test case to reproduce the issue.

Recommendation

We recommend replacing multiplication with division.

Status: Resolved

12. Last buyers will be forced to use poor exchange rates

Severity: Major

In the `buy_token` function in `contracts/token/src/contract/exec.rs:250`, when `potential_supply` exceeds `settings.allocated_supply_for_sale`, the contract automatically caps the user's `token_to_mint` to the remaining delta without any warning or refund mechanism.

This is problematic because it creates a critical economic vulnerability where late-stage buyers can lose significant funds while receiving minimal tokens. In practice, this condition occurs silently and does not refund the user any excess funds sent.

Recommendation

We recommend implementing a refund mechanism for excess funds when supply limits are reached or reverting the transaction so users can retry with a lower amount.

Status: Resolved

13. Incorrect validation of supply restriction limits

Severity: Major

In `contracts/token/src/contract/exec.rs:236`, the `buy_token` function incorrectly computes the `token_balance` variable by adding `tokens_before_fee` (token amount including fees) instead of `token_to_mint` (actual tokens received by user, excluding fees). This will cause the `SevenPercentLimitReached` validation to be incorrectly enforced, as users will be receiving `token_to_mint` instead of `tokens_before_fee` (see line `294`).

Consequently, the 7% supply restriction limit may be incorrectly triggered, preventing users from purchasing the tokens.

Recommendation

We recommend replacing `tokens_before_fee` with `token_to_mint`.

Status: Resolved

14. Updating settings on deployed contracts may fail due to gas limits

Severity: Minor

In the `update_settings` function in `contracts/deployer/src/contract/init.rs:106-150`, when updating settings with `is_apply_on_new_contracts = false`, the function attempts to update all deployed token contracts simultaneously.

This could eventually fail if there are too many contracts due to gas limits and message depth limits.

Recommendation

We recommend batching the contracts with paginations while emitting proper attributes to help the owner keep track of the upgrade progress and to know when it's complete.

Status: Resolved

15. Potential leftover funds due to imprecise liquidity splits

Severity: Minor

In `contracts/token/src/contract/reply.rs:69-84`, the `save_migration` function computes the amount of funds to distribute to the DEX contract as `convert_zig_to_uzig(half_reserve)`.

This is problematic because after sending `convert_zig_to_uzig(half_reserve)` to the creator in line `45`, the remaining uzig amount for the DEX may differ slightly due to odd-number splits. This may cause some leftover funds in the contract, which will be stuck.

Recommendation

We recommend using `convert_zig_to_uzig(remaining_reserve - half_reserve)` for the DEX transfer calculation to ensure all remaining funds are utilized.

Status: Resolved

16. Instantiate lacking proper validations

Severity: Minor

The `instantiate` function in `contracts/deployer/src/contract/init.rs:15-50` accepts all input parameters without validation, which may lead to incorrect and problematic configurations.

Specifically, the following addresses should be validated:

- `contract_owner`
- `creation_treasury`
- `trading_treasury`
- `migration_treasury`
- `dex_contract`

Additionally, it is important to ensure that numeric parameters like `curve_steepness`, `base_price`, and `target_market_cap` are within expected ranges for the bonding curve mechanism.

Recommendation

We recommend implementing parameter validation of key parameters in the `instantiate` function.

Status: Resolved

17. Ignored error during `POOL.update`

Severity: Minor

In `contracts/token/src/contract/exec.rs:301`, the return value of `POOL.update` is ignored. This is problematic because if `POOL.update` fails (e.g., due to serialization issues), the transaction will continue executing instead of reverting, potentially causing state inconsistencies.

Recommendation

We recommend propagating the error instead of ignoring it.

Status: Resolved

18. Query functions do not paginate responses

Severity: Minor

In several instances of the codebase, queries do not implement pagination when fetching values from storage. This is problematic because it could lead to performance issues and gas limit problems as the number of contracts grows. For example, the query may fail due to out-of-gas issues, causing a denial of service.

The instances are shown below:

- `get_deployed_contracts` function in `contracts/deployer/src/contract/query.rs:6-15`
- `get_token_holders` function in `contracts/token/src/contract/query.rs:31`
- `get_price_ohlc_history` function in `contracts/token/src/contract/query.rs:107`
- `get_transaction_history` function in `contracts/token/src/contract/query.rs:119`
- `get_user_balance` function in `contracts/token/src/contract/query.rs:205`

Additionally, the `get_tokens_details` query in `contracts/token/src/contract/query.rs:254` should not return the `price_ohlc_history`, as this will make the response object unpredictably large, and users should just rely on `get_price_ohlc_history` to specifically get the candle history.

Recommendation

We recommend paginating query responses and enforcing proper limits.

Status: Resolved

19. Potential query failures due to large entries in the `TRANSACTION_HISTORY`

Severity: Minor

The `get_24h_volume` function attempts to process all transactions within a 24-hour period without any size limits or pagination. As the number of transactions grows over time, this function can consume excessive gas and potentially cause out-of-gas query errors, making it impossible to retrieve volume data for active trading periods.

This is especially problematic because it recalculated all volume from the entire transaction history, so it's very inefficient and likely to cause issues as trading activity takes place.

Additionally, this issue also affects the `get_tokens_details` function in `contracts/token/src/contract/query.rs:236-241` because all entries in the `TRANSACTION_HISTORY` are iterated to compute the volume.

Recommendation

We recommend creating a state variable that can cache the 24-hour rolling trading volume rather than recalculating it ad hoc in every query.

Status: Resolved

20. Parsing failures in `calculate_price` will default to the base price

Severity: Minor

In `contracts/token/src/utils.rs:164`, the `calculate_price` function uses `Decimal::from_str(&price_rounded).unwrap_or(base_price)`. This means if parsing `price_rounded` fails, it silently defaults to `base_price`, producing a price unrelated to the actual computed value and masking the actual error.

Recommendation

We recommend propagating and returning the error.

Status: Resolved

21. The `is_apply_on_new_contracts` flag should be set to true if fee values change

Severity: Minor

In the `update_settings` function in `contracts/deployer/src/contract/init.rs:58-177`, when fee values are updated without setting `is_apply_on_new_contracts = true`, the deployer contract's fee configuration diverges from existing token contracts. This creates a critical operational issue where token contracts may attempt to send funds based on outdated fee values, potentially causing failed transactions and blocking token operations.

Recommendation

We recommend automatically setting `is_apply_on_new_contracts = true` when any fee-related values are modified to ensure immediate synchronization.

Status: Resolved

22. Conversion for the target market cap can be simplified

Severity: Informational

In `contracts/token/src/contract/query.rs:243`, the `settings.target_market_cap` field is being converted into uzig denom. This is performed by multiplying `settings.target_market_cap` with `Decimal::from_ratio(1_000_000u128, 1u128)`.

However, there is already a helper function `convert_zig_to_uzig` that encapsulates this conversion logic.

Recommendation

We recommend using the `convert_zig_to_uzig` function.

Status: Resolved

23. Funds validation can be improved

Severity: Informational

There are multiple locations in the codebase in which validation for funds can be improved to ensure that only the proper denom and amount are being sent.

For example, in `contracts/deployer/src/contract/exec.rs:56-62`, the sent funds may exceed the intended amount.

Additionally, there are multiple locations in the code where the maintainability could be improved by using the cosmwasm standard library function `must_pay`:

- `contracts/deployer/src/contract/exec.rs:51`
- `contracts/token/src/contract/exec.rs:30`
- `contracts/token/src/contract/exec.rs:174`

Recommendation

We recommend improving the funds validation as detailed above, and utilizing CosmWasm funds handling functions such as `must_pay`.

Status: Resolved

24. Functions throughout the codebase return minimal or no response attributes

Severity: Informational

Throughout the codebase, numerous functions return `Response` objects with minimal or no attributes, limiting observability and making it difficult to track and audit contract operations effectively.

The following are responses that can be improved by adding more descriptive attributes:

- `contracts/deployer/src/contract/init.rs:49`
- `contracts/deployer/src/contract/exec.rs:104`
- `contracts/token/src/contract/init.rs:48`

Recommendation

We recommend emitting detailed attributes.

Status: Acknowledged

The client states that they are intentionally conservative with event attributes. On ZigChain/CosmWasm, every attribute increases transaction size and gas, and high-churn paths (e.g., buys, mints, migrations) can cross size/gas limits or create noisy event streams that are hard to index reliably. More importantly, verbose attributes risk leaking intermediate calculations and pool state that can aid MEV/front-running. Their design treats on-chain state as the source of truth and emits only a small, stable set of attributes where they're externally actionable (e.g., method tags and final outcomes). All other details are derivable via queries (state structs, histories, and replies) without inflating logs. If stakeholders want richer telemetry, they can expose it as an opt-in (e.g., a configurable "telemetry level") so mainnet remains lean and private while testnet/ops can enable extended attributes when needed.

25. Incorrect error field returned in `extract_token_response_data`

Severity: Informational

In `contracts/deployer/src/utils.rs:74-78`, the `extract_token_response_data` function incorrectly handles the error for the missing `creator` attribute to "`_contract_address`".

This is incorrect, as the missing field in this case is "`creator`".

Recommendation

We recommend updating the field value in the error to "`creator`".

Status: Resolved

26. Usage of magic numbers

Severity: Informational

There are multiple locations in the codebase where magic numbers are used without explanation or documentation. It is best practice to utilize descriptive variables and to document these constants properly.

The following are locations where magic numbers should be replaced:

- `contracts/deployer/src/contract/exec.rs:100`
- `contracts/deployer/src/contract/reply.rs:31`
- `contracts/token/src/contract/exec.rs:331`
- `contracts/token/src/utils.rs:166`

Recommendation

We recommend replacing magic numbers with variables and code comments to explain their purpose.

Status: Resolved

Appendix

1. Test case for “Incorrect slippage tolerance logic validation”

```
#[test]
    fn test_slippage_not_working() {
        let mut deps = mock_dependencies();
        let env = mock_env();
        let (_, _, _) = setup_for_buy_token(&mut deps.as_mut());
        let mut settings = SETTINGS.load(&deps.storage).unwrap();

        // lower base price to make tokens minted large but slightly off due to
rounding
        settings.base_price = Decimal::from_ratio(1u128, 10_000u128);
        SETTINGS.save(&mut deps.storage, &settings).unwrap();

        // update pool with some circulating supply so price changes
        let mut pool = POOL.load(&deps.storage).unwrap();
        pool.circulating_supply = Uint128::new(5000);
        pool.uzigs_in_reserve = Uint128::new(1_000_000_000);
        POOL.save(&mut deps.storage, &pool).unwrap();


        let payment_amount = 1_000_000u128; // 1 uzig token in micro units
        let info = mock_info("buyer", &[cosmwasm_std::Coin::new(payment_amount,
"uzig")]);


        // set an extremely tight slippage tolerance (0.01%)
        let slippage = Some(Decimal::from_ratio(1u128, 10_000u128));


        let result = super::super::exec::buy_token(deps.as_mut(), env, info,
slippage);


        assert!(result.is_err(), "should exceed slippage");


        if let Err(err) = result {
            match err {
                Error::SlippageReached => {
                    println!("Slippage reached as expected");
                },
                _ => panic!("Unexpected error: {:?}", err),
            }
        }
}
```