MONETHIC

Razor DEX

Smart Contract Audit Report

Prepared for:
Razor DAO

Date:
03.05.2024

Version:
Final, for public release

Table of Contents

About Monethic

Monethic is a young and thriving cybersecurity company with extensive experience in various fields, including Smart Contracts, Blockchain protocols (layer 0/1/2), wallets and off-chain components audits, as well as traditional security research, starting from penetration testing services, ending at Red Team campaigns. Our team of cybersecurity experts includes experienced blockchain auditors, penetration testers, and security researchers with a deep understanding of the security risks and challenges in the rapidly evolving IT landscape. We work with a wide range of clients, including fintechs, blockchain startups, decentralized finance (DeFi) platforms, and established enterprises, to provide comprehensive security assessments that help mitigate the risks of cyberattacks, data breaches, and financial losses.

At **Monethic**, we take a collaborative approach to security assessments, working closely with our clients to understand their specific needs and tailor our assessments accordingly. Our goal is to provide actionable recommendations and insights that help our clients make informed decisions about their security posture, while minimizing the risk of security incidents and financial losses.

About Client

Razor DAO is a protocol building DeFi and infrastructure solution on Movement Labs, a network of MOVE-based blockchains. One of their first products is Razor DEX - a decentralized exchange built on x * y = k constant product model, offering services such as swaps, liquidity providing, and flash swaps.

Due to the fact that it is built on Movement, it supports both the M1 network, i.e. a chain connecting other blockchains supporting the MOVE language, and M2, i.e. first MOVE Layer-2 on Ethereum. For now, M1 supports Aptos MOVE, while M2 supports Sui MOVE.

Disclaimer

This report reflects a rigorous security assessment conducted on the specified product, utilizing industry-leading methodologies. While the service was carried out with the utmost care and proficiency, it is essential to recognize that no security verification can guarantee 100% immunity from vulnerabilities or risks.

Security is a dynamic and ever-evolving field. Even with substantial expertise, it is impossible to predict or uncover all future vulnerabilities. Regular and varied security assessments should be performed throughout the code development lifecycle, and engaging different auditors is advisable to obtain a more robust security posture.

This assessment is limited to the defined scope and does not encompass parts of the system or third-party components not explicitly included. It does not provide legal assurance of compliance with regulations or standards, and the client remains responsible for implementing recommendations and continuous security practices.

Scoping Details

The purpose of the assessment was to conduct a security assessment of the Razor DEX Smart Contracts in order to detect vulnerabilities and bad practices applied in the course of contract creation.

The report is based on the vulnerabilities found in the course of the work. The document is intended for the internal needs of Razor DAO.

The report does not take into account the vulnerabilities that arose after the test completion date.

Scope

The scope covered by the security assessment specifies that the Razor DEX contracts will be audited, the code of which has been shared on the GitHub platform with the **bff3d37ddb4e2df1a121dbd8e3b8fb19693f81ad** commit SHA hash.

No additional internal documentation was provided - only documentation on the project website was available during the audit.

The scope of testing includes the code sections listed below:

- `m1-dex/sources/*`
- `m2-dex/sources/*`

Timeframe

On April 9th Monethic was chosen as one of two providers for the security audit of Razor DEX. Work began immediately.

On April 30th, the report from the Smart Contract security assessment was delivered to the customer.

Vulnerability Classification

All vulnerabilities described in the report have been thoroughly classified in terms of the risk they generate in relation to the security of the contract implementation. Depending on where they occur, their rating can be estimated on the basis of different methodologies.

In most cases, the estimation is done by summarizing the impact of the vulnerability and its likelihood of occurrence. The table below presents a simplified risk determination model for individual calculations.

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

Vulnerabilities that do not have a direct security impact, but may affect overall code quality, as well as open doors for other potential vulnerabilities, are classified as **INFORMATIVE**.

Vulnerabilities summary

No.	Severity	Name	Status
1	Medium	Potential overflow leading to a perceived price confusion	Acknowledged
2	Medium	One step admin address change	Resolved
3	Medium	Validation of m1-dex init_module function caller is not performed	Resolved
4	Low	Comments assumptions are not always fulfilled	Resolved
5	Low	Invalid test cases	Resolved
6	Low	Coins comparison is not always performed	Acknowledged
7	Low	Fee boundaries are not enforced	Resolved
8	Low	Assert in swap_balance_for_balance always returns True	Resolved
9	Low	Functions for config setting could be called while contract is paused	Resolved
10	Low	Usage of default values for pair LP name and symbol might be problematic	Partially Resolved
11	Informative	Potential underflow in mint function	Resolved
12	Informative	Invalid coding pattern	Resolved

Technical summary

1. Potential overflow leading to a perceived price confusion

Severity

Medium

Location

`m1-dex/sources/swap_library.move:104-114`
`m2-dex/sources/swap_library.move:70-80`

Description

It was observed that the `RazorSwapPool` modules in `m1-dex` and `m2-dex` implement an `overflow_add` function. This function implements an addition that allows overflow. Although this function is not used to calculate actual prices during swaps, it is used to update the persistent storage of the pool. Namely, the `update_internal` functions use it to calculate the values of `last_price_x_cumulative` and `last_price_y_cumulative`. Those values are then used in the emitted events. Those events are observed by interested parties (users, for instance), who might base their actions on the received event data. Furthermore, the m1-dex implements a `get_last_price_cumulative` function, which returns said values for other contracts. Although it is not likely for these values to actually overflow in the production environment, should this situation occur, the impact on the protocol might be devastating as users and potentially other contracts may attempt to use the protocol and expect significantly different results.

```
/// Add but allow overflow
public fun overflow_add(a: u128, b: u128): u128 {
    let r = MAX_U128 - b;
    if (r < a) {
        return a - r - 1
    };
    r = MAX_U128 - a;
    if (r < b) {
        return b - r - 1
    };
    a + b
}
```

Remediation

Although overflow sometimes has a valid business reason to be possible, it is recommended to prevent the possibility of overflow for any finance-related values. Should the overflow be deemed required in this case, the protocol should also indicate when it happened so that interested parties are notified about every aspect of this calculation.

Status: Acknowledged

2. One step admin address change

Severity

Medium

Location

`m1-dex/sources/swap_library.move:615`
`m2-dex/sources/swap_library.move:806`

Description

The `m1-dex` and `m2-dex` contracts differ in terms of determining the administrator. In `m1`, the administrator is global and can manage the entire protocol. In `m2`, the administrator is defined as per pool. The `set_admin_address` function is used to change the administrator's address.

It was found that for both DEXs, the administrator change functionality is accomplished by calling a single function. This is problematic because if a mistake is made, for example, specifying an address with a typo when calling a function from the script, or the newly assigned administrator needs to be recalled at the last minute - it will not be possible to regain control over the protocol or pool.

```
    public entry fun set_admin_address(
        account: &signer,
        admin_address: address
    ) acquires AdminData {
        let admin_data =
borrow_global_mut<AdminData>(RESOURCE_ACCOUNT_ADDRESS);
        assert!(signer::address_of(account) == admin_data.admin_address,
ERR_FORBIDDEN);
        admin_data.admin_address = admin_address;
    }
```

Remediation

We recommend changing the administrator's address in two steps. His address should be proposed as a future administrator, and through the `claim` function and the implemented access control mechanism - the future administrator, proving that he has been correctly indicated and is aware of the granted rights, should receive them by performing another transaction in the protocol. This way, it will be impossible to make a mistake when changing the administrator, because the "old" administrator will be able to revoke the transfer of rights at any time.

Status: Resolved

3. Validation of `m1-dex` `init_module` function caller is not performed

Severity

Medium

Location

`m1-dex/sources/swap.move:173`

Description

It has been noticed that the function responsible for initializing the `RazorSwapPool` module, `init_module`, does not have validation implemented whether it has been called by `RESOURCE_ACCOUNT_ADDRESS`. It should be invoked only by an authorized entity, in this case - `@razor`.

```
    // initialize
    fun init_module(admin: &signer) {
        // init admin data
        let signer_cap =
resource_account::retrieve_resource_account_cap(admin, DEV);
        let resource_account =
account::create_signer_with_capability(&signer_cap);
        move_to(&resource_account, AdminData {
            signer_cap,
            dao_fee_to: DEPLOYER_ADDRESS,
            admin_address: DEPLOYER_ADDRESS,
            dao_fee: 5,          // 1/6 to dao fee
            swap_fee: 30,        // 0.3%
            dao_fee_on: true,   // default true
            is_pause: false,     // default false
        });
        // init pair info
        move_to(&resource_account, PairInfo {
            pair_list: vector::empty(),
        });
    }
```

Remediation

We recommend implementing caller validation of the init_module function so that only `RESOURCE_ACCOUNT_ADDRESS` is authorized to this operation.

Status: Resolved

---

4. Comments assumptions are not always fulfilled

Severity

LOW

Location

`m1-dex/sources/swap.move:197`

Description

The `get_reserves_size` function is used to return the reserves of two assets - X and Y. Before the implementation of the function, there is a comment indicating that it always returns `(X_reserve, Y_reserve)`.

After analyzing the code, it was found that this is not true. By using the compare function, it is checked whether `X < Y`, and if it returns False, i.e. `X > Y`, the result of the function is `(Y_reserve, X_reserve)`. This is therefore counterintuitive and may be problematic in functionalities that use get_reserves_size, as the reserves returned by it may be swapped.

```
    /// get reserves size
    /// always return (X_reserve, Y_reserve)
    public fun get_reserves_size<X, Y>(): (u64, u64) acquires LiquidityPool {
        if (RazorPoolLibrary::compare<X, Y>()) {
            let lp = borrow_global<LiquidityPool<X,
Y>>(RESOURCE_ACCOUNT_ADDRESS);
            (coin::value(&lp.coin_x_reserve), coin::value(&lp.coin_y_reserve))
        } else {
            let lp = borrow_global<LiquidityPool<Y,
X>>(RESOURCE_ACCOUNT_ADDRESS);
            (coin::value(&lp.coin_y_reserve), coin::value(&lp.coin_x_reserve))
        }
    }
```

Remediation

We suggest adjusting the logic of the function so that the returned values are consistent with those expected by functionalities that call `get_reserves_size` in their code.

Status: Resolved

Description of `get_reserves_size` was changed, so now it may return different values order, depending on the X and Y comparison.

---

5. Invalid test cases

Severity

LOW

Location

`m2-dex/sources/swap.move:966-1516`

Description

It was observed that the `m2-dex` contract defines a test module containing tests for a different project. Namely, the test cases are related to the `animeswap` project and do not test the functions defined in the `m2-dex` contract. As a consequence, `m2-dex` is not covered by tests, and therefore it is not possible to verify whether the functionalities work as intended.

```
#[test_only]
module defi::animeswap_tests {
    use sui::coin::{mint_for_testing as mint, burn_for_testing as burn};
    use sui::coin::{Self, Coin};
    use sui::test_scenario::{Self as test, Scenario, next_tx, ctx};
    use defi::animeswap::{Self, LiquidityPools, LPCoin};
    use sui::clock::{Self, Clock};
    // use std::debug;
    const TEST_ERROR: u64 = 10000;

[..]
```

Remediation

It is recommended to write valid tests for the functionalities implemented in the `m2-dex` project.

Status: Resolved

6. Coins comparison is not always performed

Severity

LOW

Location

`m2-dex/sources/swap.move:349`

Description

While analyzing the contracts, it was noticed that in `m2-dex` in the `remove_liquidity_entry` function, for which there is a requirement that `X < Y`, comparison of these two coins is not performed.

Consequently, in most cases, it will be impossible to execute the transaction if this condition is not met - because the pair will not be found.

```
    /// remove liqudity entry function
    /// require X < Y
    public entry fun remove_liquidity_entry<X, Y>(
        lps: &mut LiquidityPools,
        clock: &Clock,
        liquidity: Coin<LPCoin<X, Y>>,
        liquidity_desired: u64,
        amount_x_min: u64,
        amount_y_min: u64,
        ctx: &mut TxContext,
    ) {
        let (coin_x, coin_y) = remove_liquidity<X, Y>(
            lps,
            clock,
            liquidity,
            liquidity_desired,
            amount_x_min,
            amount_y_min,
            ctx,
        );
        transfer::public_transfer(coin_x, tx_context::sender(ctx));
        transfer::public_transfer(coin_y, tx_context::sender(ctx));
    }
```

Remediation

We suggest that, as with other functions, you call coin sorting using the `compare()` function.

Status: Acknowledged

---

7. Fee boundaries are not enforced

Severity

LOW

Location

`m2-dex/sources/swap.move:815-836`

Description

It was noticed that in `m2-dex`, when determining the fee amount, no upper value limits were introduced, forcing the privileged user to change the protocol parameters to adapt to the top-down security and boundaries imposed. Due to the fact that in `m2-dex` the `dao_fee` and `swap_fee` parameters are of type `u64`, their maximum values are very high.

Occurrences of no limits:

- `set_dao_fee`
- `set_swap_fee`

```
    public entry fun set_dao_fee(
        lps: &mut LiquidityPools,
        dao_fee: u64,
        ctx: &mut TxContext,
    ) {
        assert!(lps.admin_data.admin_address == tx_context::sender(ctx),
ERR_FORBIDDEN);
        if (dao_fee == 0) {
            lps.admin_data.dao_fee_on = false;
        } else {
            lps.admin_data.dao_fee_on = true;
            lps.admin_data.dao_fee = dao_fee;
        };
    }
    public entry fun set_swap_fee(
        lps: &mut LiquidityPools,
        swap_fee: u64,
        ctx: &mut TxContext,
    ) {
        assert!(lps.admin_data.admin_address == tx_context::sender(ctx),
ERR_FORBIDDEN);
        lps.admin_data.swap_fee = swap_fee;
    }
```

Remediation

We suggest, similarly to `m1-dex`, the use of upper limits for which it will be possible to increase the fee, for example 10%, which will effectively protect users against setting too high and dangerous levels of fees.

Status: Resolved

---

8. Assert in swap_balance_for_balance always returns True

Severity

LOW

Location

`m2-dex/sources/swap.move:573`

Description

Inside the `swap_balance_for_balance` function called during coins swapping, it was found that one of the asserts always returns `True` due to incorrect use of logical operators. Consequently, it is unable to check the logic for which it was created.

Current code version:

```
assert!((amount_x_in > 0 && amount_y_in == 0) || (amount_x_in == 0 ||
amount_x_in > 0), ERR_INPUT_VALUE);
```

Probable target code version:

```
assert!((amount_x_in > 0 && amount_y_in == 0) || (amount_x_in == 0 &&
amount_y_in > 0), ERR_INPUT_VALUE);
```

Remediation

We suggest changing the logical operators and replacing the `amount_x_in` parameter with `amount_y_in` in assertion.

Status: Resolved

---

9. Functions for config setting could be called while contract is paused

Severity

LOW

Location

`m2-dex/sources/swap.move:573`

Description

The `m1-dex` and `m2-dex` contracts have a pause mechanism, used in emergency situations. It stops most functionalities, allowing the owner to analyze a potential incident or perform other time-consuming activities.

According to the comment in the `pause` function, the only operation allowed if the contract is paused is an execution of LP removal operation. However, it was found that it is also possible to change the configuration of fees, administrator, or withdrawal of dao fee. This goes against the logic of the contract pause and may have negative consequences for users.

Occurrences:

- `set_dai_fee_to`
- `set_admin_address`
- `set_dao_fee`
- `set_swap_fee`
- `withdraw_dao_fee`

Remediation

We suggest that you clearly describe which features are subject to pause and which are not in your technical documentation. Additionally, we recommend analyzing whether the above-mentioned functions should have an implemented contract status verification mechanism.

Status: Resolved

The comment next to the `pause` function now says that it only applies to user-accessible operations - the administrator still has full rights to use his privileged functions.

---

10. Usage of default values for pair LP name and symbol might be problematic

Severity

LOW

Location

`m1-dex/sources/swap.move:707, 714`

Description

When creating a pair in `m1-dex`, its name is created from a string consisting of the string `"Razor-"` and the elements of the pair - `X` and `Y` tokens. However, if the name length exceeds 32, the default name `"Razor DEX LPs"` is set.

However, for the symbol, the hardcoded value `"Razor-LP"` is set each time, without the possibility of editing. This is problematic because in the case of several LPs with default names, they will differ only in the tokens they consist of. As for LP Coins - all of them will have the same name, these may or may not be a business premise.

```
if (string::length(&lp_name) > MAX_COIN_NAME_LENGTH) {
            lp_name = string::utf8(b"Razor DEX LPs");
        };
        // create lp coin
        let (lp_b, lp_f, lp_m) = coin::initialize<LPCoin<X, Y>>(
            &resource_account_signer,
            lp_name,
            string::utf8(b"Razor-LP"),
            true
        );
```

Remediation

We suggest that if `lp_name` exceeds `MAX_COIN_NAME_LENGTH`, an error is returned, not the default value. Additionally, we recommend analyzing whether the name of the LP Coin symbol should not include the components of the pair to which it is assigned.

Status: Partially Resolved

The symbols of individual LPs have been changed correctly, but if the name of the pool itself is too long - the default one, "Razor DEX LPs", will be selected.

---

11. Potential underflow in `mint` function

Severity

INFORMATIVE

Location

`m1-dex/sources/swap.move:839`
`m2-dex/sources/swap.move:644`

Description

It was noticed that in the mint function, both in `m1-dex` and `m2-dex`, if `total_supply` is `zero`, `liquidity` is calculated according to the formula: `sqrt(amount_x, amount_y) - MINIMUM_LIQUIDITY`. This is problematic because nowhere is it verified whether the square root of the multiplication of two amounts is greater than or equal to `MINIMUM_LIQUIDITY`.

As a consequence, an underflow scenario is possible, but due to the fact that MOVE by design protects against overflow or underflow, it does not pose a significant threat. However, it may be problematic from a functional point of view, as calling such a function will result in failure.

```
if (total_supply == 0) {
            liquidity = sqrt(amount_x, amount_y) - MINIMUM_LIQUIDITY;
```

Remediation

We suggest verifying that `sqrt(amount_x, amount_y)` is greater than or equal to `MINIMUM_LIQUIDITY` value. If so, an appropriate message should be returned to the caller.

Status: Resolved

---

12. Invalid coding pattern

Severity

INFORMATIVE

Location

`m2-dex/sources/swap_library.move:114-115`

Description

The `m2-dex` contract defines a `compare` function used to ensure proper ordering of the Coins. However, the function is designed to return early, and should it reach the end, it is supposed to return an error and terminate the execution while reverting storage. The current code that is used for this works as expected, however is somewhat obfuscated and can be simplified.

```
    // compare type, when use, CoinType1 should < CoinType2
    public fun compare<X, Y>(): bool {
        let type_name_x = type_name::into_string(type_name::get<X>());
        let type_name_y = type_name::into_string(type_name::get<Y>());
        let length_x = ascii::length(&type_name_x);
        let length_y = ascii::length(&type_name_y);
        let bytes_x = ascii::into_bytes(type_name_x);
        let bytes_y = ascii::into_bytes(type_name_y);
        if (length_x < length_y) return true;
        if (length_x > length_y) return false;
        let idx = 0;
        while (idx < length_x) {
            let byte_x = *vector::borrow(&bytes_x, idx);
            let byte_y = *vector::borrow(&bytes_y, idx);
            if (byte_x < byte_y) {
                return true
            } else if (byte_x > byte_y) {
                return false
            };
            idx = idx + 1;
        };
        assert!(false, ERR_COIN_TYPE_SAME_ERROR);
        false
    }
```

Remediation

It is recommended to replace the assert and return value statements with an: `abort ERR_COINT_TYPE_SAME_ERROR` statement.

Status: Resolved

END OF THE REPORT