Security Review Report
NM-0060: mySwap Braavos (Layer 2)

NETHERMIND

(Oct 21, 2022)

Contents

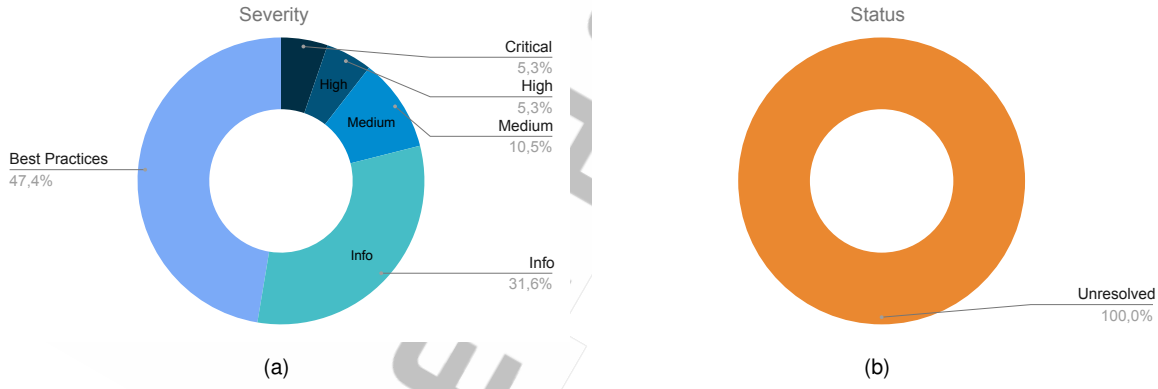
1 Executive Summary

This document presents the security review performed by Nethermind on the Layer 2 contracts of mySwap Braavos written in Cairo code. The protocol is based on the Uniswap V2 Curve, with some design changes in terms of a) pools centralization; b) funds centralization; c) use of the proxy pattern; d) possibility of setting arbitrary fees by the pool creator.

Pools centralization: the AMM contract is responsible for creating and managing all the pools. When a new pool is created, only the token's liquidity contract is deployed, but the pool is virtual and exists only in the context of the AMM contract as a record in storage variables. This gives the AMM contract full control over all the pools and makes the AMM contract the main interaction layer for users. Thus, if the AMM is hacked, funds from all pools can be lost; **i) Funds centralization**: the AMM contract is a vault for all tokens present in the `mySwap` protocol. Also the AMM is the owner of all liquidity tokens for each pool; **Proxy**: the AMM contract is implemented as a proxy. The `proxy_admin` role is held by one single address. **ii) Pool creators can set arbitrary fees**: differently from `Uniswap V2` that limits the fees to predefined values, `mySwap` does not check the fee for valid ranges. It is possible to set any arbitrary value for the fee up to 100% (and above, since the constructor does not validate the `fee` for range; **iii) Missing expiration deadline for transactions**: Unlike `Uniswap V2`, `mySwap` does not implement an expiration date for transactions. **iv) The concentration of power into one single entity** (`proxy_admin`) and one single contract holding the funds of several pools represents a risk to mySwap.

The documentation provided in the README.md details the four most important use-cases of the protocol: **a) create new pool**: Creates a new pool from a token pair, in case one does not exist. Accepts the two token addresses, initial balance for each, and the pool fee. During the pool creation, a new token is minted, and the bulk of the minted tokens are assigned to the caller. A small amount of 1,000 tokens is deducted from the pool creator's share, and remains in hold of the AMM contract itself; **b) swap**: Implements the swap functionality. The allowance for the AMM should be approved beforehand; **c) add liquidity**: Adds liquidity to an existing pool. Parameters include the two tokens, with desired and minimum amounts for each. Both token amounts should be approved to the AMM contract; **d) withdraw liquidity**: Implements the liquidity withdrawal from the pool. The liquidity token should be approved to the AMM beforehand. Although the documentation was sufficient for the audit, we evaluate the documentation as `low` for not presenting the differences between the `Uniswap V2` and `mySwap`, which can lead a non-technical user to assume that both protocols work exactly just the same.

The codebase is composed of 1,481 lines of Cairo code. **During the initial audit, we point out 20 potential issues**. The most important issues are related to: a) denial of service attack; b) unsafe change of proxy implementation; c) missing expiration deadline for transactions; d) arbitrary fees can be set by pool creators. **We were not able to execute the test suite during the initial audit**. The distribution of issues is summarized in the figure below. **This document is organized as follows.** Section 2 presents the files in the scope of this audit. Section 3 summarizes the findings in a table. Section 4 discusses the risk rating methodology adopted for this audit. Section 5 details each finding. Section 6 discusses the documentation provided for this audit. Section 7 presents the output of the automated test suite. Section 8 concludes the audit report.



Distribution of issues: Critical (1), **High** (1), **Medium** (2), **Low** (0), **Undetermined** (0), **Informational** (6), **Best Practices** (9).
Distribution of status: Unresolved (19)

Summary of the Audit

Audit Type	Security Review
Initial Report	Oct. 21, 2022
Response from Client	-
Final Report	-
Methods	Manual Review, Automated Analysis
Repository	mySwap-Braavos
Commit Hash (Initial Audit)	8c1cd6c9c3356f3f519e8765e8bcc543cbf004d0
Documentation	README.md
Documentation Assessment	Low
Test Suite Assessment	Unable to execute

2 Contracts

	Contract	Lines of Code	Lines of Comments	Comments Ratio	Blank Lines	Total Lines
1	src/openzeppelin/upgrades/library.cairo	69	23	33.3%	21	113
2	src/openzeppelin/utils/constants.cairo	11	18	163.6%	12	41
3	src/openzeppelin/access/ownable.cairo	51	20	39.2%	16	87
4	src/openzeppelin/token/erc20/interfaces/IERC20.cairo	23	2	8.7%	11	36
5	src/openzeppelin/security/safemath.cairo	87	15	17.2%	12	114
6	src/myswap/Proxy.cairo	53	9	17.0%	12	74
7	src/myswap/library.cairo	683	289	42.3%	186	1158
8	src/myswap/Amm.cairo	118	5	4.2%	13	136
9	src/myswap/token/erc20/library.cairo	239	33	13.8%	61	333
10	src/myswap/token/erc20/ERC20_Lp.cairo	120	16	13.3%	23	159
11	src/myswap/interfaces/IERC20_Lp.cairo	27	3	11.1%	13	43
	Total	**1481**	**433**	**29.2%**	**380**	**2294**

3 Summary of Issues

	Finding	Severity	Update
1	Pool creators can set arbitrary fees and block the whole protocol	Critical	Unresolved
2	Functions not following the Checks-Effects-Interactions pattern	High	Unresolved
3	Centralization of power in the `admin`	Medium	Unresolved
4	Unsafe change of proxy implementation hash	Medium	Unresolved
5	LP token with variable number of decimals	Info	Unresolved
6	More than one pool can receive the same name	Info	Unresolved
7	No possibility of changing the `proxy_admin`	Info	Unresolved
8	Unnecessary `shifts` in function `calc_a_div_b_times_c(...)`	Info	Unresolved
9	Unused parameter in function `create_new_pool()`	Info	Unresolved
10	Using `POOL_UPPER_BOUND_L -1` in function `calc_amount_out(...)`	Info	Unresolved
11	Function `update_pool_reserves(...)` can be refactored	Best Practices	Unresolved
12	Functions missing the clause `with_attr error_message(...)`	Best Practices	Unresolved
13	Functions not validating the `pool_id` for existence	Best Practices	Unresolved
14	Hardcoded value not assigned to a constant	Best Practices	Unresolved
15	Interface does not match the implementation	Best Practices	Unresolved
16	Missing events emission in `myswap/library.cairo`	Best Practices	Unresolved
17	Some external functions are unusable	Best Practices	Unresolved
18	Unchecked `transfer` operations	Best Practices	Unresolved
19	Unused members in the `Pool` struct	Best Practices	Unresolved

4 Risk Rating Methodology

The risk rating methodology used by Nethermind follows the principles established by the OWASP Foundation. The severity of each finding is determined by two factors: **Likelihood** and **Impact**.

Likelihood is a measure of how likely the finding is to be uncovered and exploited by an attacker. This factor will be one of the following values:

a) **High**: The issue is trivial to exploit and has no specific conditions that need to be met;

b) **Medium**: The issue is moderately complex and may have some conditions that need to be met;

c) **Low**: The issue is very complex and requires very specific conditions to be met.

When defining the likelihood of a finding other factors are also considered. These can include but are not limited to: Motive, opportunity, exploit accessibility, ease of discovery and ease of exploit.

Impact is a measure of the damage that may be caused if the finding were to be exploited by an attacker. This factor will be one of the following values:

a) **High**: The issue can cause significant damage such as loss of funds or the protocol entering an unrecoverable state;

b) **Medium**: The issue can cause moderate damage such as impacts that only affect a small group of users or only a particular part of the protocol;

c) **Low**: The issue can cause little to no damage such as bugs that are easily recoverable or cause unexpected interactions that cause minor inconveniences.

When defining the impact of a finding other factors are also considered. These can include but are not limited to: Data/state integrity, loss of availability, financial loss, reputation damage. After defining the likelihood and impact of an issue, the severity can be determined according to the table below.

		Severity Risk		
Impact	**High**	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Info/Best Practices	Low	Medium
	Undetermined	Undetermined	Undetermined	Undetermined
		Low	**Medium**	**High**
		Likelihood		

To address issues that do not fit a High/Medium/Low severity, Nethermind also uses three more finding severities: **Informational**, **Best Practices**, and **Undetermined**.

a) **Informational** findings do not pose any risk to the application, but they carry some information that the audit team intends to formally pass to the client;

b) **Best Practice** findings are used when some piece of code does not conform with smart contract development best practices;

c) **Undetermined** findings are used when we cannot predict the impact or likelihood of the issue.

5 Issues

5.1 General

5.1.1 [Medium] Centralization of power in the `admin`

File(s): `mySwap`

Description: The `mySwap` AMM protocol has some design features that combined together can lead to some potential risks. The features are: **a) pools centralization:** The `AMM` contract is responsible for creating and managing all the pools. When a new pool is created, only the token's liquidity contract is deployed, but the pool is virtual and exists only in the context of the `AMM` contract as a record in storage variables. This gives the `AMM` contract full control over all the pools and makes the `AMM` contract the main interaction layer for the users. Thus, if the `AMM` is hacked, funds from all pools can be stolen; **b) funds centralization:** The `AMM` contract is a vault for all tokens present in a `mySwap` protocol. Also the `AMM` is the owner of all `LP tokens` for each pool; **c) use of proxy pattern:** The `AMM` contract is implemented as a proxy. The `proxy_admin` role is held by a single address. All the control over the funds and the pools is held by one single address, which is a single point of failure for the whole protocol.

Recommendation(s): Adopt some defensive strategies for the protocol, such as: **a) consider using the Factory Pattern adopted in Uniswap V2**: one Factory contract is responsible for creating the Pair contracts, which hold the funds. Each pair contract is also an LP token; **b) separate the core AMM functionality from the user interface layer**: Uniswap V2 does that by introducing the Router contract, which allows users a safe interaction with the protocol; **c) proxy**: The use of the proxy pattern may bring additional attack vectors and vulnerabilities to the protocol. The proxy pattern may be secured by: i) using the freeze period for changing the `proxy_admin` address or implementation contract hash; ii) using the propose-accept pattern; iii) using the multi-signature pattern to distribute the `proxy_admin` role.

Status: Unresolved

Update from the client:

5.1.2 [Best Practices] Hardcoded value not assigned to a constant

File(s): `mySwap`

Description: The value `1,000` is hardcoded, instead of being assigned to a constant.

Recommendation(s): Assign hardcoded values to constants following the best software engineering practices.

Status: Unresolved

Update from the client:

5.2 `src/myswap/interfaces/IERC20_Lp.cairo`

5.2.1 [Best Practices] Interface does not match the implementation

File(s): `src/myswap/interfaces/IERC20_Lp.cairo`

Description: The `IERC20_Lp` interface does not contain functions: `owner(...)`, `increaseAllowance(...)`, `decreaseAllowance(...)`, `transferOwnership(...)` and `renounceOwnership(...)`. Those functions are implemented in `ERC20_Lp.cairo` as either `@view` or `@external`.

Recommendation(s): Consider unifying the implementation with the interface for the `ERC20_Lp` token.

Status: Unresolved

Update from the client:

5.3 `src/myswap/Amm.cairo`

5.3.1 [Medium] Unsafe change of proxy implementation hash

File(s): `src/myswap/Amm.cairo`

Description: The change of the implementation contract hash with function `upgrade(...)` is secured by asserting that only the `proxy_admin` can perform this action. The upgrade logic is in the implementation contract. If an incorrect hash is provided during the update, the upgrade logic may be lost. In this scenario, the contract funds will be inaccessible (lost).

Recommendation(s): To mitigate such a scenario, additional safety mechanisms can be implemented. We recommend moving the upgrade logic from the implementation contract to the proxy contract. Moreover, implement securing mechanisms for changing the implementation hash. Examples of such mechanisms are: **a) proposal-acceptance pattern**: changes are proposed and they must be accepted by other trusted actors; **b) multi-signature**: multiple actors must provide the signature to validate the change.

Status: Unresolved

Update from the client:

5.4 src/myswap/library.cairo

5.4.1 [Critical] Pool creators can set arbitrary fees and block the whole protocol

File(s): src/myswap/library.cairo

Description: The current design allows for the creation of pools with arbitrary (and possibly high) fees. Also, the application only allows for the creation of one single pool for a given pair of tokens. The fee value of a pool can't be changed. **Thus, a malicious user can block a token from being used on `mySwap` by creating pools having this token and high fees. The protocol is also susceptible to be blocked for all tokens if a malicious user decides to create all possible combinations of pools setting a high value for the fee. High fees are likely to discourage users from using the pool. If the fee is set above `100%`, all swap operations are blocked**. The total freedom of selecting the fees can also bring other attack vectors.

Recommendation(s): We recommend: a) predefine a valid set of fees or impose a range for fees; b) define each pool using the address of each token plus the fee.

Status: Unresolved

Update from the client:

5.4.2 [High] Functions not following the Checks-Effects-Interactions pattern

File(s): src/myswap/library.cairo

Description: The functions `create_new_pool(...)`, `add_liquidity(...)`, and `swap(...)` are not following the Checks-Effects-Interactions (CEI) pattern, and are susceptible to reentrancy. This would allow users to take advantage of pools composed by `ERC-777` tokens (or any other token with hooks).

Recommendation: Use the Checks-Effects-Interactions (CEI) pattern or locks for making your functions non reentrant.

Status: Unresolved

Update from the client:

5.4.3 [Info] LP token with variable number of decimals

File(s): src/myswap/library.cairo

Description: The function `create_new_pool(...)` creates a new token for representing the liquidity provided. The new token is created with a number of decimals equal to the average number of decimals of the tokens composing the pool. This may create confusion, since each pair LP token can potentially have a different number of decimals. It is a good practice to keep decimal numbers between LP tokens unified.

Recommendation(s): This design decision can be rediscussed by the development team. The liquidity token may be created with a constant number, to avoid confusion. Common choice for decimal numbers is `18`.

Status: Unresolved

Update from the client:

5.4.4 [Info] More than one pool can receive the same name

File(s): src/myswap/library.cairo

Description: The function `create_new_pool(...)` generates pool names based on the name of each token composing the pair. A malicious user can create a new token mimicking some well-known token and create a second pair having the pool name. Since the pair of tokens will have different addresses, the pool will be created and swaps will occur. The users will see two pools with the same name and they can add liquidity to the malicious pool. This situation can compromise the reputation of the protocol.

Recommendation(s): Do not allow more than one pool having the same name in order to avoid scam pools. Consider adding a list of tokens in your frontend which is vetted from your end as a best effort to eliminate the possibility of users trading with scam tokens. Consider

Status: Unresolved

Update from the client:

5.4.5 [Info] Unnecessary `shifts` in function `calc_a_div_b_times_c(...)`

File(s): src/myswap/library.cairo

Description: The function `calc_a_div_b_times_c(...)` makes a shift left followed by a shift right. This use case is unclear for us. Although we have no evidence that something is wrong, additional shift operations are redundant and do not add additional precision. Below we reproduced the code snippet.

```
func calc_a_div_b_times_c{...}(a: Uint256, b: Uint256, c: Uint256) -> (r: Uint256) {
    assert a.high = 0;
    assert b.high = 0;
    assert c.high = 0;

    ####################################################################
    # @audit-issue revert if "b" is zero.
    # @audit       Why not do normal division without shifting?
    ####################################################################

    # a*c should be less than 2**256
    let (mul, _) = uint256_mul(a, c);
    if (mul.high == 0) {
        let (shl_mul) = uint256_shl(mul, Uint256(128, 0));  # we know the results should be lt 256 bits
        let (quotient, _) = uint256_unsigned_div_rem(shl_mul, b);
        let (res) = uint256_shr(quotient, Uint256(128, 0));  # in practice it is quotient.high
        return (res,);
    } else {
        # mul > 128 bits
        let (quotient, _) = uint256_unsigned_div_rem(mul, b);  # reminder is irrelevant
        assert quotient.high = 0;  # otherwise, we get amounts higher than 128-bits
        return (quotient,);
    }
}
```

Recommendation(s): Double check this code.

Status: Unresolved

Update from the client:

5.4.6 [Info] Unused parameter in function `create_new_pool()`

File(s): src/myswap/library.cairo

Description: The `pool_name` parameter is not used inside the `create_new_pool()`.

Recommendation(s): Remove the unused parameter.

Status: Unresolved

Update from the client:

5.4.7 [Info] Using `POOL_UPPER_BOUND_L - 1` in function `calc_amount_out(...)`

File(s): src/myswap/library.cairo

Description: The function `calc_amount_out(...)` is using `POOL_UPPER_BOUND_L - 1` to check for overflow. The proper check seems to be `uint256_le(sum, Uint256(POOL_UPPER_BOUND_L, POOL_UPPER_BOUND_H))`. The code snippet is shown below.

```
func calc_amount_out{...}(
    pool_id: felt, token_from_addr: felt, token_to_addr: felt, amount_from: Uint256
) -> (amount_to: Uint256) {
    ...
    // Verify amounts did not overflow
    let (sum: Uint256, _) = uint256_add(amm_from_balance, amount_from);
    ##############################################################################
    # @audit-issue Using "POOL_UPPER_BOUND_L - 1" instead of "POOL_UPPER_BOUND_L"
    ##############################################################################
    let (no_overflow: felt) = uint256_le(
        sum, Uint256(POOL_UPPER_BOUND_L - 1, POOL_UPPER_BOUND_H)
    );
    assert_not_zero(no_overflow);

    return (amount_to,);
}
```

Recommendation(s): Double check this since other parts of the code are checking for overflow using the command `uint256_le(sum, Uint256(POOL_UPPER_BOUND_L, POOL_UPPER_BOUND_H))`.

Status: Unresolved

Update from the client:

5.4.8 [Best Practices] Function `update_pool_reserves(...)` can be refactored

File(s): src/myswap/library.cairo

Description: The `update_pool_reserves(...)` function is called by `withdraw_liquidity(...)` and `add_liquidity(...)`. It's a good practice to write reusable code. However, the function `update_pool_reserves(...)` has specific operations and assertions target to withdrawals (and not for the other operations). This increases the effort on Program Comprehension. The code fragment is shown below.

```
func update_pool_reserves{...}(
        pool_id: felt, a_delta: Uint256, b_delta: Uint256
    ) {
    alloc_locals;

    // get pool reserves
    let (resa, resb) = get_pool_reserves(pool_id);

    // Calc new reserve
    let (new_a_reserves, carry) = uint256_add(resa, a_delta);
    let (is_val_nn) = uint256_signed_nn(a_delta);
    let carry = carry * is_val_nn;
    with_attr error_message("mySwap: reserves underflow 1") {
        assert_not_zero(1 - carry);
    }
    let (new_b_reserves, carry) = uint256_add(resb, b_delta);
    let (is_val_nn) = uint256_signed_nn(b_delta);
    let carry = carry * is_val_nn;
    with_attr error_message("mySwap: reserves underflow 2") {
        assert_not_zero(1 - carry);
    }
    ...
}
```

Recommendation(s): Apply the Extract Method refactoring. This method is a technique to separate a method by extracting some code from an existing method as a new method. The purpose of this refactoring is to improve the comprehension of the program by splitting a method that is too long, or a method that implements multiple features into each one. For more information, please check this reference.

Status: Unresolved

Update from the client:

5.4.9 [Best Practices] Functions missing the clause `with_attr error_message(...)`

File(s): src/myswap/library.cairo

Description: Some validations are missing the clausule `with_attr error_message(...)`. The places and functions are listed below.

```
func swap{...}(
    pool_id: felt, token_from_addr: felt, amount_from: Uint256, amount_to_min: Uint256
) -> (amount_to: Uint256) {
    ...
    ############################################################################
    # @audit Missing with_attr error_message(...)
    ############################################################################
    assert_not_zero(pool_id);

    let (n) = num_of_pools.read();
    ############################################################################
    # @audit Missing with_attr error_message(...)
    ############################################################################
    assert_nn_le(pool_id, n);
    ...
    ############################################################################
    # @audit Missing with_attr error_message(...)
    ############################################################################
    assert (token_from_addr - pool.token_a_address) * (token_from_addr - pool.token_b_address) = 0;
    ...
}
```

```
func calc_amount_out{...}(
    pool_id: felt, token_from_addr: felt, token_to_addr: felt, amount_from: Uint256
) -> (amount_to: Uint256) {
    ...
    ############################################################################
    # @audit Missing with_attr error_message(...)
    ############################################################################
    assert_not_zero(amount_from.low + amount_from.high);
    ...
    ############################################################################
    # @audit Missing with_attr error_message(...)
    ############################################################################
    assert_nn_le(carry, 0);
    ...
    ############################################################################
    # @audit Missing with_attr error_message(...)
    ############################################################################
    let (is_eq_0: felt) = uint256_eq(numerator_h, Uint256(0, 0));
    ...
    ############################################################################
    # @audit Missing with_attr error_message(...)
    ############################################################################
    assert_not_zero(is_eq_0);
    ...
    ############################################################################
    # @audit Missing with_attr error_message(...)
    ############################################################################
    assert_not_zero(amount_to.low + amount_to.high);
    ...
    ############################################################################
    # @audit Missing with_attr error_message(...)
    ############################################################################
    assert_not_zero(no_overflow);
    ...
}
```

```
func calc_optimal_quote{...}(
    pool_id: felt,
    token1_addr: felt,
    token1_amount: Uint256,
    token2_addr: felt,
    token2_amount: Uint256,
) -> (actual1: Uint256, actual2: Uint256) {
    ...
    ############################################################################
    # @audit Missing with_attr error_message(...)
    ############################################################################
    assert_lt_felt(token1_addr, token2_addr);
    ...
    ############################################################################
    # @audit Missing with_attr error_message(...)
    ############################################################################
    assert_not_zero(is_enough_balance);
    ...
    ############################################################################
    # @audit Missing with_attr error_message(...)
    ############################################################################
    assert_not_zero(is_enough_balance);
    ...
    ############################################################################
    # @audit Missing with_attr error_message(...)
    ############################################################################
    assert_not_zero(flag1 + flag2);
    ...
}
```

```
func get_opposite_token{...}(
    pool_id: felt, token_address: felt
) -> (op_token_address: felt) {
    let (pool) = amm_pools.read(id=pool_id);

    if (token_address == pool.token_a_address) {
        return (pool.token_b_address,);
    } else {
        ########################################################################
        # @audit Missing with_attr error_message(...)
        ########################################################################
        assert token_address = pool.token_b_address;
        return (pool.token_a_address,);
    }
}
```

Recommendation(s): Follow the Cairo best programming practice of enclosing validations with the clausule `with_attr error_message(...)`.

Status: Unresolved

Update from the client:

5.4.10 [Best Practices] Functions not validating the `pool_id` for existence

File(s): `src/myswap/library.cairo`

Description: The functions listed below are not validating the `pool_id` for existence.

```
// The LP shares should be divided by the pool's total shares in order to get the LP percentage in the pool
func get_lp_balance{...}(
    pool_id: felt, lp_address: felt
) -> (shares: Uint256) {
    ########################################################
    # @audit-issue Not checking "pool_id" for existence
    ########################################################
    let (pool) = amm_pools.read(id=pool_id);
    let (token_lp) = IERC20_Lp.balanceOf(contract_address=pool.liq_token, account=lp_address);
    return (token_lp,);
}
```

```
func get_total_shares{...}(
    pool_id: felt
) -> (total_shares: Uint256) {
    ########################################################
    # @audit-issue Not checking "pool_id" for existence
    ########################################################
    let (pool) = amm_pools.read(id=pool_id);
    let (ts) = IERC20_Lp.totalSupply(contract_address=pool.liq_token);
    return (ts,);
}
```

```
func get_pool_reserves{...}(
    pool_id: felt
) -> (tokena_reserves: Uint256, tokenb_reserves: Uint256) {
    alloc_locals;

    ##################################################################
    # @audit-issue Function must revert if "pool_id" is invalid
    ##################################################################
    let (resa) = per_pool_token_a_reserves.read(pool_id);
    let (resb) = per_pool_token_b_reserves.read(pool_id);

    return (tokena_reserves=resa, tokenb_reserves=resb);
}
```

Recommendation(s): Always validate the `pool_id` for existence.

Status: Unresolved

Update from the client:

5.4.11 [Best Practices] Missing events emission in `myswap/library.cairo`

File(s): `src/myswap/library.cairo`

Description: The functions below should emit events.

```
- func create_new_pool(...)
- func add_liquidity(...)
- func withdraw_liquidity(...)
- func swap(...)
- func update_pool_reserves(...)
- _deploy_liq_token(...)
```

Recommendation(s): As a good practice, smart contracts should emit events in state transitions, initialize functions and constructors. Events also help with post-deployment management.

Status: Unresolved

Update from the client:

5.4.12 [Best Practices] Unchecked `transfer` operations

File(s): `src/myswap/library.cairo`

Description: The `success` value on token transfer is not checked. This may lead to incompatibility of `mySwap` protocol with `ERC20` tokens that incorrectly return `FALSE` value on failure, but do not revert. Such a situation may result in a security issue, where the token transfer would fail, but the `mySwap` protocol wouldn't react to that accordingly.

Recommendation(s): Check the return value for all transfer operations.

Status: Unresolved

Update from the client:

5.4.13 [Best Practices] Unused members in the `Pool` struct

File(s): `src/myswap/library.cairo`

Description: The `Pool` struct contains unused members: `cfmm_type`, `token_a_reserves` and`token_b_reserves`. Values of those fields are never read. The reserve values of each token in each pool are stored in global variables `per_pool_token_a_reserves` and `per_pool_token_b_reserves`. Doubling variables for reserve storage reduces the code readability and may create accidental errors during further development, e.g. updating `Pool.token_a_reserves` instead of `per_pool_token_a_reserves` or not updating `per_pool_token_a_reserves` and `per_pool_token_b_reserves` in new implementation contracts. The struct and the redundant storage variables are shown below.

```
// Define a Pool struct
struct Pool {
    // Pool name (str_to_felt)
    name: felt,
    // Token 1 ERC-20 contract address
    token_a_address: felt,
    // Token 1 liquidity
    token_a_reserves: Uint256,
    // Token 2 ERC-20 contract address
    token_b_address: felt,
    // Token 2 liquidity
    token_b_reserves: Uint256,
    // Fee as 10^(-3) percent (1000 == 1%, 100 == 0.1%, 10 = 0.01%)
    fee_percentage: felt,
    // Constant Function Market Maker Invariant (0=constant product, 1=stable-swap)
    cfmm_type: felt,
    // Liquidity Token address
    liq_token: felt,
}

// per-pool token_a reserves
@storage_var
func per_pool_token_a_reserves(pool_id: felt) -> (amount: Uint256) {
}

// per-pool token_b reserves
@storage_var
func per_pool_token_b_reserves(pool_id: felt) -> (amount: Uint256) {
}
```

Recommendation(s): Remove the unused members of the `Pool` struct.

Status: Unresolved

Update from the client:

5.5 `src/myswap/Proxy.cairo`

5.5.1 [Info] No possibility of changing the `proxy_admin`

File(s): `src/myswap/Proxy.cairo`

Description: Currently, `proxy_admin` address is only defined during the proxy contract deployment. However, changing the `proxy_admin` may be necessary in the future.

Recommendation(s): Consider implementing a secure `proxy_admin` update utilizing safe change patterns, such as: **a) proposal-acceptance pattern:** changes are proposed and to finalize must be accepted by other trusted actors; **b) multi-signature:** to validate the change multiple actors must provide the signature.

Status: Unresolved

Update from the client:

5.6 src/myswap/token/erc20/ERC20_Lp.cairo

5.6.1 [Best Practices] Some external functions are unusable

File(s): src/myswap/token/erc20/ERC20_Lp.cairo

Description: Liquidity token contract `ERC20_Lp` contains external functions for transferring ownership `transferOwnership(...)` and for renouncing ownership `renounceOwnership(...)`. Those functions must be called by the owner of the contract. However, the owner of every liquidity token is the `AMM` contract. The `AMM` contract does not contain the functionality to call `transferOwnership(...)` and `renounceOwnership(...)`, which makes those functions non-callable.

Recommendation(s): Consider removing non-callable functions `transferOwnership(...)` and `renounceOwnership(...)` or implement calling those functions from the `AMM` contract.

Status: Unresolved

Update from the client:

6 Documentation Evaluation

Technical documentation is created to explain what the software product does. This way, developers and stakeholders can easily follow the purpose and the underlying functionality of each file/function/line. Documentation can come not only in the form of a `README.md` but also using code as documentation (to write clear code), diagrams, websites, research papers, videos and external documentation. Besides being a good programming practice, proper technical documentation improves the efficiency of audits. Less time can be spent understanding the protocol and more time can be put towards auditing which improves the efficiency and overall output of the audit. The README.md presents the core use-cases, but does not highlight the differences between the `Uniswap V2` implementation and `mySwap`. The documentation can be improved for a deeper comprehension of the protocol. **The code presents a ratio of 29.2% lines of comments per line of code**. It could benefit from following an inline documentation standard, such as SIMP or even `NatSpec`.

7 Test Suite Evaluation

7.1 Contracts Compilation

The contracts compiled with just a few warning, as presented below.

```
./compile.sh
~/cairo/NM-0060/myswap_audit-8c1cd6c9c3356f3f519e8765e8bcc543cbf004d0/src
↪  ~/cairo/NM-0060/myswap_audit-8c1cd6c9c3356f3f519e8765e8bcc543cbf004d0
starknet-compile myswap/Amm.cairo --output ../artifacts/Amm.json --abi ../artifacts/abis/Amm.json
myswap/Amm.cairo:2:1: Unexpected token Token('SLASH', '/'). Expected one of: "%builtins", "%lang", "@", "alloc_locals",
↪  "assert", "call", "const", "dw", "from", "func", ...
// %builtins pedersen range_check bitwise
^
starknet-compile myswap/Proxy.cairo --output ../artifacts/Proxy.json --abi ../artifacts/abis/Proxy.json
myswap/Proxy.cairo:1:1: Unexpected token Token('SLASH', '/'). Expected one of: "%builtins", "%lang", "@",
↪  "alloc_locals", "assert", "call", "const", "dw", "from", "func", ...
// SPDX-License-Identifier: MIT
^
starknet-compile myswap/token/erc20/ERC20_Lp.cairo --output ../artifacts/ERC20_Lp.json --abi
↪  ../artifacts/abis/ERC20_Lp.json
myswap/token/erc20/ERC20_Lp.cairo:1:1: Unexpected token Token('SLASH', '/'). Expected one of: "%builtins", "%lang",
↪  "@", "alloc_locals", "assert", "call", "const", "dw", "from", "func", ...
// SPDX-License-Identifier: MIT
^
~/cairo/NM-0060/myswap_audit-8c1cd6c9c3356f3f519e8765e8bcc543cbf004d0
```

7.2 Tests Output

The tests did not run. The output is reproduced below.

```
$ pytest
============================= test session starts ==============================
platform linux -- Python 3.8.10, pytest-7.1.2, pluggy-1.0.0
rootdir: /home/cris/cairo/NM-0060/myswap_audit-8c1cd6c9c3356f3f519e8765e8bcc543cbf004d0
plugins: web3-5.30.0, typeguard-2.13.3, asyncio-0.19.0
asyncio: mode=strict
collected 0 items / 1 error

==================================== ERRORS ====================================
_____________________ ERROR collecting tests/test_amm.py ______________________
ImportError while importing test module
↪  '/home/cris/cairo/NM-0060/myswap_audit-8c1cd6c9c3356f3f519e8765e8bcc543cbf004d0/tests/test_amm.py'.
Hint: make sure your test modules/packages have valid Python names.
Traceback:
/usr/lib/python3.8/importlib/__init__.py:127: in import_module
    return _bootstrap._gcd_import(name[level:], package, level)
/home/cris/cairo/NM-0060/myswap_audit-8c1cd6c9c3356f3f519e8765e8bcc543cbf004d0/myswap_audit-pytests/tests/test_amm.py:12:
↪  in <module>
    ???
tests/utils/utils.py:12: in <module>
    from starkware.starknet.business_logic.transaction.objects import InternalInvokeFunction
E   ModuleNotFoundError: No module named 'starkware.starknet.business_logic.transaction'
============================== warnings summary ================================
../../../cairo_venv/lib/python3.8/site-packages/frozendict/__init__.py:16
  /home/cris/cairo_venv/lib/python3.8/site-packages/frozendict/__init__.py:16: DeprecationWarning: Using or importing
  ↪  the ABCs from 'collections' instead of from 'collections.abc' is deprecated since Python 3.3, and in 3.10 it will
  ↪  stop working
    class frozendict(collections.Mapping):

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
=========================== short test summary info ============================
ERROR tests/test_amm.py
!!!!!!!!!!!!!!!!!!!! Interrupted: 1 error during collection !!!!!!!!!!!!!!!!!!!!
======================== 1 warning, 1 error in 1.05s ===========================
```

8 About Nethermind

Founded in 2017 by a small team of world-class technologists, Nethermind builds Ethereum solutions for developers and enterprises. Boosted by a grant from the Ethereum Foundation in August 2018, our team has worked tirelessly to deliver the fastest Ethereum client in the market. Our flagship Ethereum client is all about performance and flexibility. Built on .NET core, a widespread, enterprise-friendly platform, Nethermind makes integration with existing infrastructures simple, without losing sight of stability, reliability, data integrity, and security

Nethermind is made up of several engineering teams across various disciplines, all collaborating to realize the Ethereum roadmap, by conducting research and building high-quality tools. Teams focus on specific areas of the Ethereum problem space. Each consists of specialists and experienced developers working alongside interns, learning the ropes in the Nethermind Internship Program.

Our mission is to gather passionate talent from around the world, and to tackle some of the blockchain's most complex problems. Nethermind provides software solutions and services for developers and enterprises building the Ethereum ecosystem. We offer security reviews to projects built on EVM compatible chains and StarkNet. We have expertise in multiple areas of the Ethereum ecosystem, including protocol design, smart contracts (written in Solidity and Cairo), MEV, etc. We develop some of the most used tools on Starknet and one of the most used Ethereum clients. Learn more about us at https://nethermind.io.

Disclaimer

This report is based on the scope of materials and documentation provided by you to Nethermind in order that Nethermind could conduct the security review outlined in **1. Executive Summary** and **2. Audited Files**. The results set out in this report may not be complete nor inclusive of all vulnerabilities. Nethermind has provided the review and this report on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Blockchain technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. This report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on this report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, Nethermind disclaims any liability in connection with this report, its content, and any related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. Nethermind does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and Nethermind will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.