Blaize.Security

October 20th, 2023 / V. 2.0

StarkDefi.

STARKDEFI

SMART CONTRACT AUDIT

TABLE OF CONTENTS

AUDIT RATING

SCORE **9.75** /10

The scope of the project includes StarkDefi:

1st iteration:
factory.cairo
pair.cairo
router.cairo
utils.cairo

2nd iteration:
factory.cairo
Pair.cairo
pairFees.cairo
router.cairo

Repository: https://github.com/Starkdefi/StarkDefi

1st iteration:
Branch: cairo1-upgrade
Initial commit:

- 47451aaf5974e9b6059a4e8327afec902873d2f2

Final commit:

- 47451aaf5974e9b6059a4e8327afec902873d2f2

2nd iteration:
Branch: feat/stable-swap
Initial commit:

- 1beb3bcf526e4f9c7e150146a8a3757a5c04fc95

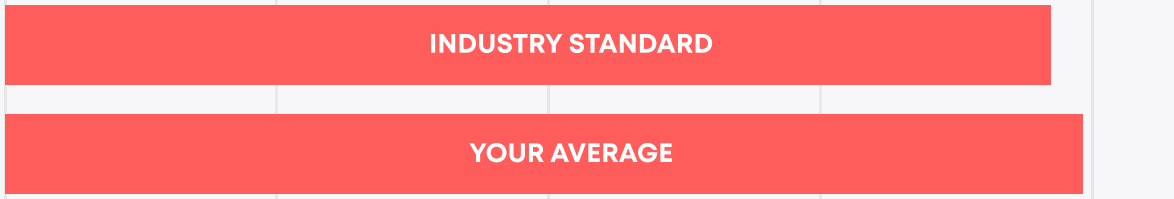
Final commit:

- 30f6a24c4b9fe27fdc7e2c9bde9e4e785d56783f

TECHNICAL SUMMARY

During the audit, we examined the security of smart contracts for the StarkDefi protocol. Our task was to find and describe any security issues in the smart contracts of the platform. This report presents the findings of the security audit of the **StarkDefi** smart contracts conducted: **1st** iteration between **September 19th, 2023** and **September 29th, 2023** and **2nd** iteration between **October 4th, 2023** and **October 20th, 2023**

Testable code



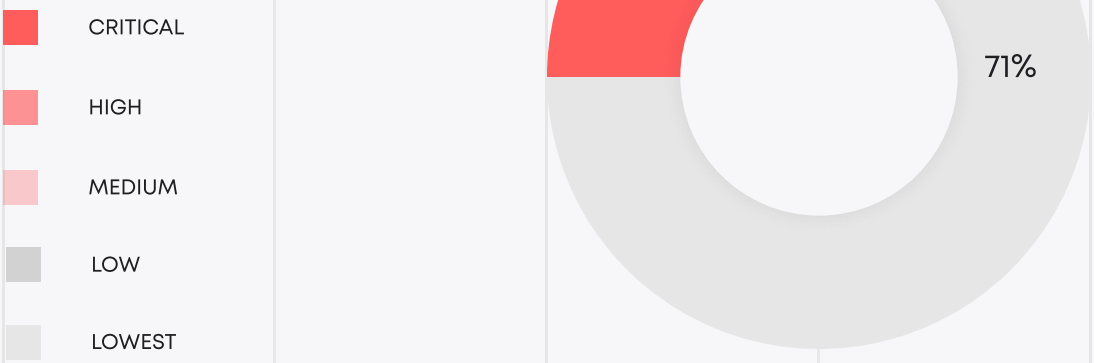
Auditors approved code as testable within the industry standard.

The audit scope includes all tests and scripts, documentation, and requirements presented by the **StarkDefi** team. The tests are prepared based on the set of Scarb framework tests and scripts from additional testing strategies, and includes testable code from manual and exploratory rounds.

However, to ensure the security of the contract, the **Blaize.Security** team suggests that the **StarkDefi** team follow post-audit steps:

1. launch **active protection** over the deployed contracts to have a system of early detection and alerts for malicious activity. We recommend the AI-powered threat prevention platform **VigiLens**, by the **CyVers** team.
2. launch a **bug bounty program** to encourage further active analysis of the smart contracts.

THE GRAPH OF VULNERABILITIES DISTRIBUTION:



The table below shows the number of the detected issues and their severity. A total of 14 problems were found. 14 issues were fixed or verified by the StarkDefi team.

	FOUND	FIXED/VERIFIED
Critical	2	2
High	1	1
Medium	1	1
Low	0	0
Lowest	10	10

SEVERITY DEFINITION

Critical

The system contains several issues ranked as very serious and dangerous for users and the secure work of the system. Requires immediate fixes and a further check.

High

The system contains a couple of serious issues, which lead to unreliable work of the system and migh cause a huge data or financial leak. Requires immediate fixes and a further check.

Medium

The system contains issues that may lead to medium financial loss or users' private information leak. Requires immediate fixes and a further check.

Low

The system contains several risks ranked as relatively small with the low impact on the users' information and financial security. Requires fixes.

Lowest

The system does not contain any issues critical to the secure work of the system, yet is relevant for best practices

AUDITING STRATEGY AND TECHNIQUES APPLIED/PROCEDURE

Blaize.Security auditors start the audit by developing an **auditing strategy** - an individual plan where the team plans methods, techniques, approaches for the audited components. That includes a list of activities:

Manual audit stage

- Manual line-by-line code by at least 2 security auditors with crosschecks and validation from the security lead;
- Protocol decomposition and components analysis with building an interaction scheme, depicting internal flows between the components and sequence diagrams;
- Business logic inspection for potential loopholes, deadlocks, backdoors;
- Math operations and calculations analysis, formula modeling;
- Access control review, roles structure, analysis of user and admin capabilities and behavior;
- Review of dependencies, 3rd parties, and integrations;
- Review with automated tools and static analysis;
- Vulnerabilities analysis against several checklists, including internal Blaize.Security checklist;
- Storage usage review;
- Gas (or tx weight or cross-contract calls or another analog) optimization;
- Code quality, documentation, and consistency review.

For advanced components:

- Cryptographical elements and keys storage/usage audit (if applicable);
- Review against OWASP recommendations (if applicable);
- Blockchain interacting components and transactions flow (if applicable);
- Review against CCSSA (C4) checklist and recommendations (if applicable);

Testing stage:

- Development of edge cases based on manual stage results for false positives validation;
- Integration tests for checking connections with 3rd parties;
- Manual exploratory tests over the locally deployed protocol;
- Checking the existing set of tests and performing additional unit testing;
- Fuzzy and mutation tests (by request or necessity);
- End-to-end testing of complex systems;

In case of any issues found during audit activities, the team provides detailed recommendations for all findings.

EXECUTIVE SUMMARY

Blaize Security team has received smart contracts written in Cairo programming language and designed for the Starknet network. Contracts represent a DEX with an Automated Market Maker algorithm.

The audit's goal was to analyze the safety of smart contracts. The process of audit included both manual audit and intensive testing stage. Thus, the audit included:

- Verification of the correctness of AMM invariant and correctness of reserves changes;
- Verification of fee calculation and distribution;
- Checking the flow of volatile or stable pair creation, adding/removing liquidity, and exchanging tokens, including ensuring common DEX protection measures such as slippage protection and deadline of any action such as exchange of tokens;
- Analysis of smart contracts against the list of common Cairo vulnerabilities, including access control protection measures;
- A comprehensive unit testing of smart contracts, including edge-case scenarios for invariant, slippage protection, different amounts of tokens, and flashloan attacks.

During the first audit iteration, auditors detected no critical issues. All identified issues were informational in nature, related to lack of documentation in the code, absence of error messages, verification of the desired logic of the contracts, and compatibility with the latest version of tools.

As for the second iteration, the StarkDeFi team upgraded the protocol with a stables swap and an updated fee system. During the second iteration of the audit, the team identified and resolved several critical issues: invalid invariant k calculation and tokens swap path array handling. Patching also included a high-risk issue related to the user's problem of collecting fees after withdrawing liquidity. Several other informational issues were also identified and resolved.

Also, the latest upgrade turned contracts into upgradeable ones, with several setters for critical values added. Auditors initially recommended such an update as the one believed to be necessary. However, the update has inadvertently complicated the situation and increased security risks for the protocol. Therefore, auditors removed the issue and recommendation from the report and advised the StarkDefi team to revert the functionality. Such changes bring additional risks of centralization and single-point-of-failure, as in the case of unauthorized access or loss of access to the admin account, the protocol will have a threat of irreversible upgrades with a threat for kept funds.

The client has acknowledged these potential areas of concern and presented plans for future improvements. They intend to maintain the router's address in future versions, allow users to select a fee level when creating a pair, and make necessary upgrades to the factory. They also plan to manage contract mutations better and use a multisig as the fee handler once launched. These future improvements, once implemented, are expected to enhance the security and functionality of the smart contracts. Despite the acknowledgement, contracts fail the access control and potential backdoors checks, thus the rating is decreased. Though, the existing strategy of StarkDeFi team “softens” the effect.

Besides that, the overall security of smart contracts is high. Contracts are well-written and follow the best practices of Cairo. The StarkDefi team prepared a sufficient set of unit tests. The Blaize Security team conducted an intensive testing stage with a focus on the swap and fee calculation logic, including checks of the slippage protection, AMM implementation (k invariant), flashloan manipulations (influence of swaps with big liquidity) and testing with different values of tokens (to check the stability of implemented invariant). The protocol has passed all the scenario tests.

	RATING
Security	9.5
Logic optimization	9.9
Code quality	9.8
Test coverage	9.8
Total	9.75

PROTOCOL OVERVIEW

DESCRIPTION

1. StarkDPair:
- This contract implements a stable or volatile pair using the uniV2 (x*y=k) volatile pair and Solidly stableswap (x*y(x^2 + y^2) =k) curve.
- It includes storage structs for configuration, pair information, global fees accumulation, and relative fees accumulation.

2. PairFees:
- This contract stores the fees collected by the StarkDefi Pair.
- It includes storage structs for the pair, token0, token1, factory, and protocol fees.
- The contract includes functions for claiming LP fees, getting protocol fees, updating protocol fees, and claiming protocol fees.

3. StarkDFactory:
- This contract is the StarkDefi Pair Factory responsible for creating pairs and setting fees.
- It includes storage structs for configuration, fees, pairs, valid pairs, and all pairs.

4. StarkDRouter Contract:
- This contract serves as a router for swapping tokens.
- It interacts with the factory contract to retrieve pair addresses and handles the swapping of tokens between different pairs.

ROLES AND RESPONSIBILITIES

factory.cairo

1. fee_handler:
- Responsible for setting and updating fees pairs.
- Has the authority to set the fee address and fee-handler address.
- Can set custom fees for specific pairs.

2. fee_to:
- Receives the fees collected by the pairs.
- Can be set by the fee handler.

pairFees.cairo

1. pair:
- Role represents the corresponding pair contract for which the fees are being collected. The pair contract is responsible for interacting with the PairFees contract and performing fee-related operations.
- The PairFees contract allows only the pair contract to call certain functions, ensuring that only the authorized pair contract can claim LP fees and update protocol fees.

LIST OF VALUABLE ASSETS

Pair token - token which is sent to users who provide liquidity to the pair.
Token0 - first underlying token of a created pair.
Token1 - second underlying token of a created pair.

SETTINGS

factory.cairo

1. fee_to address: The address where the fees collected by the pair contract are transferred. Can be changed with set_fee_to()
2. fee_handler address: The address that has the authority to set fees and manage the factory contract. Can be changed with set_fee_handler()
3. fees.stable and fees.volatile: The fees charged for each pair, both stable and volatile. Can be changed with set_fee()
4. pair_info.custom_fee: Custom fees set for specific pairs. Can be changed with set_custom_pair_fee()
5. pair_class_hash: This setting represents the class hash of the pair contract. It is stored in the `Config` struct in the `StarkDFactory` contract and is used when creating a new pair. Cannot be changed.
6. vault_class_hash: This setting represents the class hash of the fee vault contract. It is stored in the `Config` struct in the `StarkDFactory` contract and is used when creating a new pair. Cannot be changed

Pair.cairo

1. token0 and token1: These settings represent the addresses of the tokens used in the pairs. They can be configured when creating a pair and are stored in the `Config` struct in the `StarkDPair` contract.
2. _factory: factory contract address that is set when deploying a contract. Cannot be changed.
3. _entry_locked: boolean expression to prevent reentrancy. Sets false when contract deployed. When a function is invoked, it is set to true at the start, and the function returns false at the end.
4. vault_class_hash: In the `constructor` function of the `StarkDPair` contract, the `vault_class_hash` is passed as an argument and stored in the `config` struct. It is used later in the `_initialise_fee_vault` internal function to deploy the fee vault contract.
5. stable: the `stable` setting is part of the `Config` struct and is set during the contract's constructor. It indicates whether the pair follows the stableswap curve (`x*y(x^2 + y^2) = k`) or the uniV2 curve (`x*y = k`).

router.cairo

1. _factory: factory contract address that is set when deploying a contract. Cannot be changed.

pairFees.cairo

1. pair : The pair address is a crucial setting in the PairFees contract. It represents the address of the corresponding pair contract for which the fees are being collected. Cannot be changed.

2. token0 and token1: The token addresses are the addresses of the tokens involved in the pair contract. They determine the tokens for which fees are being collected. Cannot be changed.

3. factory: It represents the address of the factory contract that manages the creation and management of pair contracts. Cannot be changed

DEPLOYMENT SCRIPT

As for StarkNet, contracts deployed using CLI commands. StarkDefi team is using **sncast** CLI tool to deploy contracts. More information can be found on https://foundry-rs.github.io/starknet-foundry/starknet/index.html.

LIST OF POTENTIAL THREATS

Unauthorized Access to Swaps/Liquidity

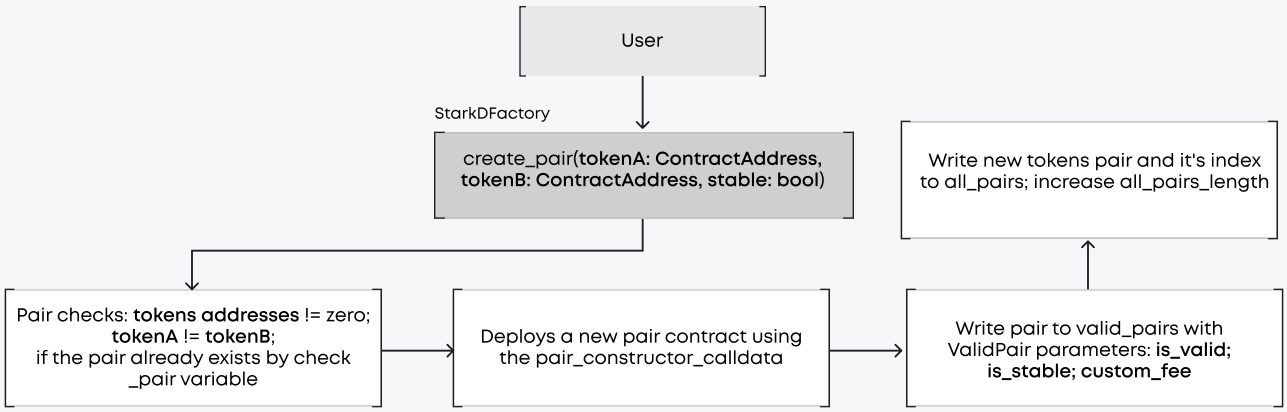
Added critical values' setters in the smart contracts increase the risk of unauthorized/accidental access to all swaps/liquidity if someone can manipulate the classhash. Such a potential backdoor poses a significant threat to the security of the smart contracts, potentially compromising their integrity and the safety of the assets they manage. Though it needs to be mentioned, that while this is valid for Pairs created after a new pair classhash is set in the factory, it doesn't affect previously deployed Pairs as the Pair contract itself is not upgradable. So the potential risk of loss or liquidity will affect Pairs deployed with the malicious classhash

Complex Logic and Additional Risks from Contract Upgradability

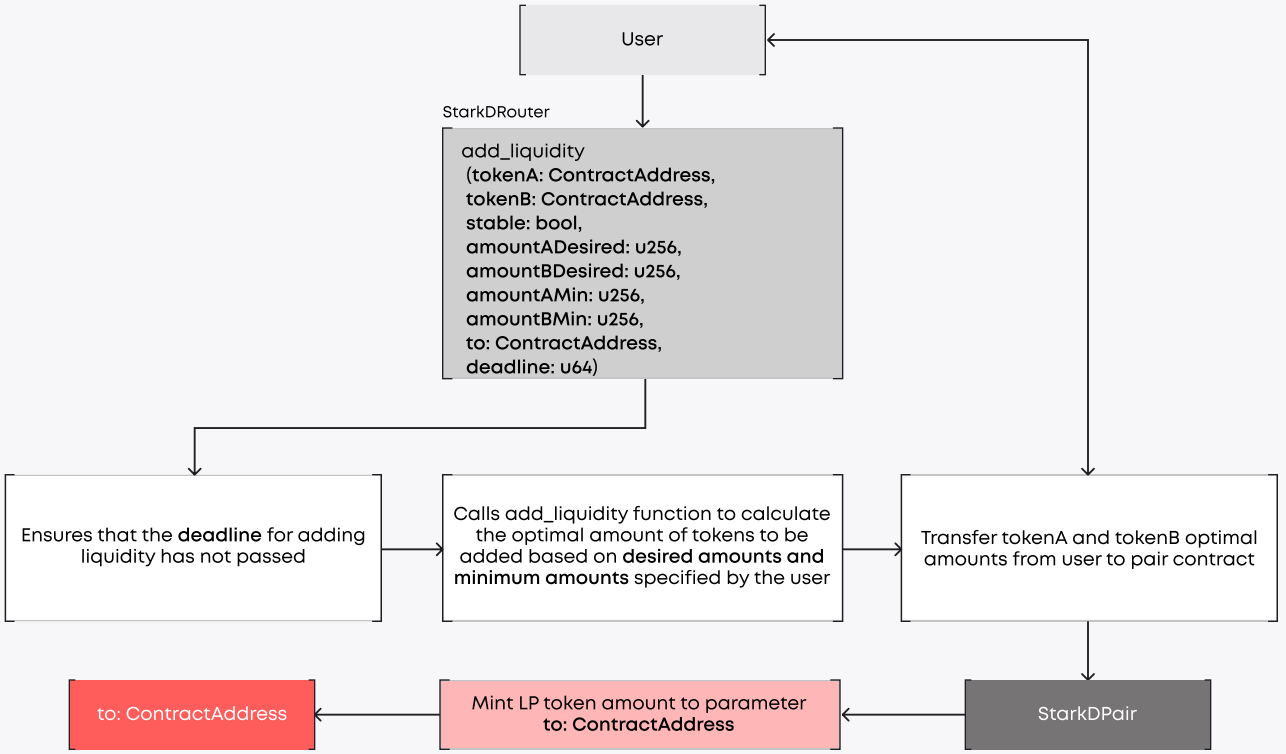
The upgradability of the contracts, while initially seen as a beneficial feature, introduces complex logic that could potentially create additional risks. This could lead to unforeseen issues and vulnerabilities, threatening the smart contracts' stability and security. Such includes a risk of single-point-of-failure, when the unauthorized upgrade may prevent further control over the protocol and bring threat to the kept funds.

S T A R K D E F I

Creating Liquidity Pairs

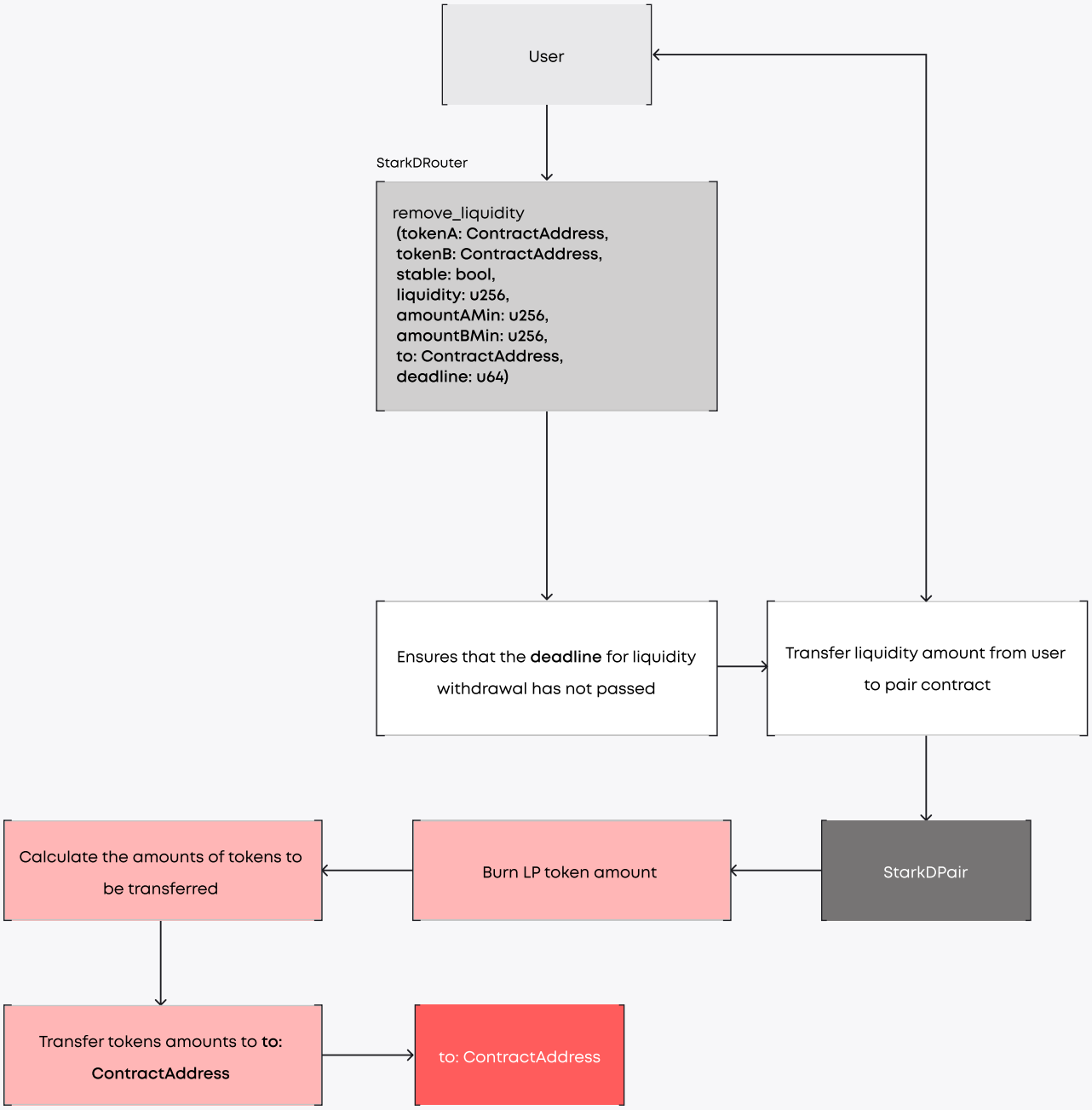


Adding Liquidity



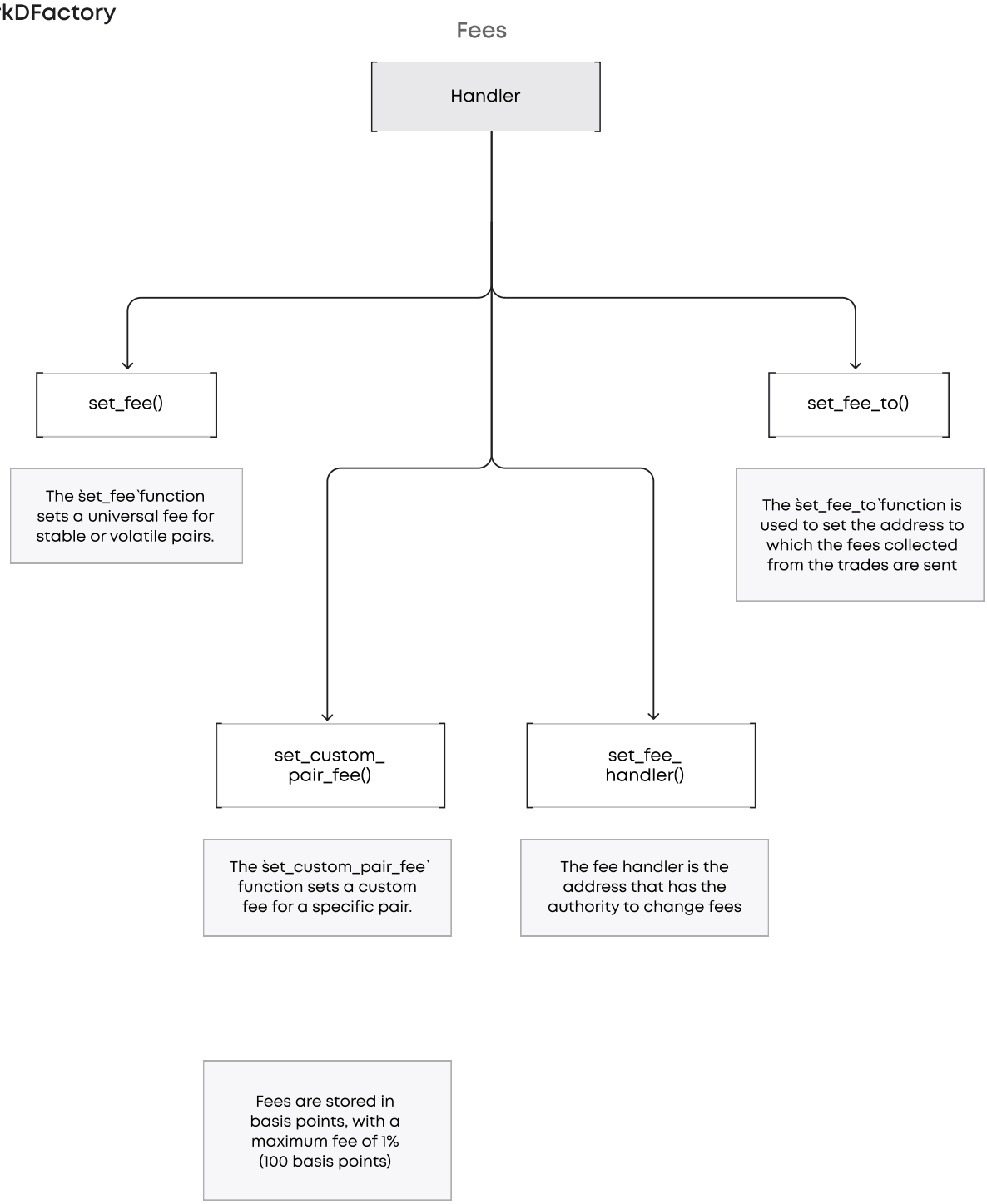
S T A R K D E F I

Liquidity Withdrawal



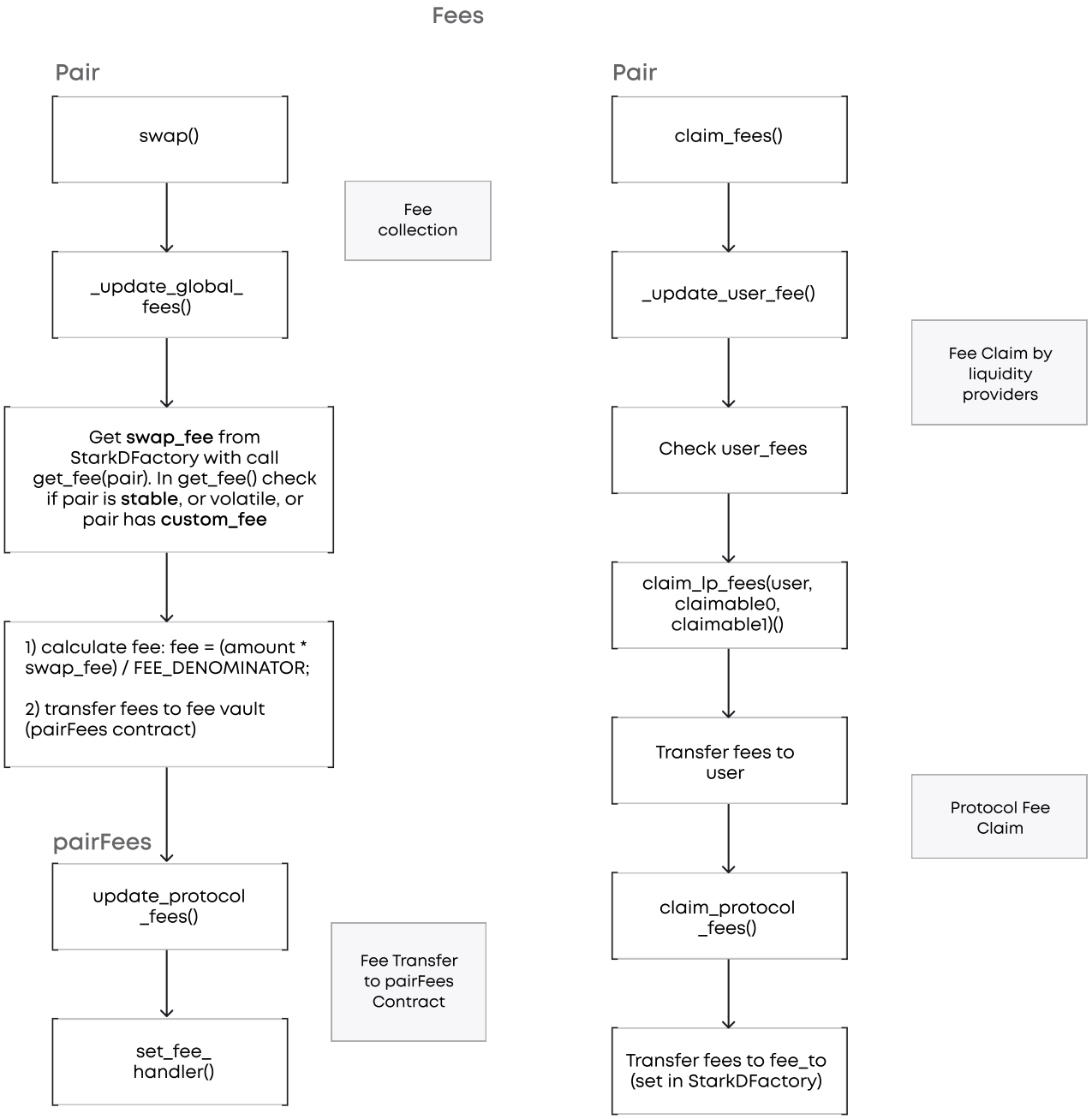
STARKDEFI

StarkDFactory



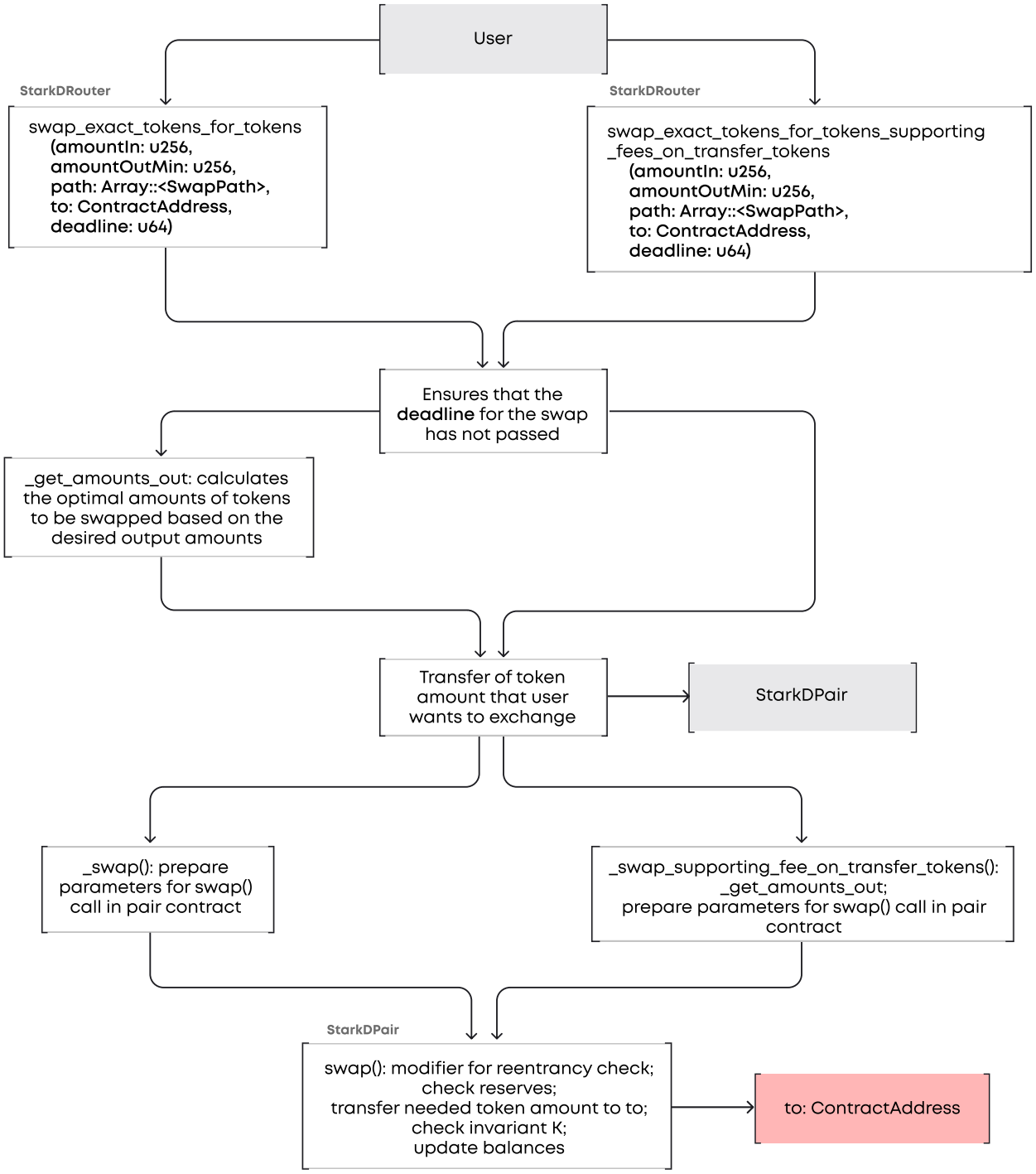
S T A R K D E F I

Pair and pairFees



STARKDEFI

Swap



COMPLETE ANALYSIS (1ST ITERATION)

LOWEST-1	✔ Resolved

Lack of comments and documentation.

It is highly recommended to add natspec comments to the code and general architecture scheme into the readme or docs section of the repository. For now, all contracts lack comments and documentation, making it challenging to provide a quick dive into the purpose and functionality of the code.

Recommendation:

Add comments and documentation to the code to improve readability and maintainability.

Post-audit:

Comments and documentation were added.

LOWEST-2	✔ Resolved

Lack of error messages in assert statements.

StarkDRouter: _add_liquidity(), line 256.
Assert statement in the contract does not include an error message. This could make it difficult to debug and understand the cause of any failed assertions.

Recommendation:

Include descriptive error message in assert statement to aid in debugging and understanding the code.

Post-audit:

Descriptive error message was included.

LOWEST-3	✔ Verified

Contracts are not compiled using the latest Scarb version.

Currently, contracts are compiled fine using Scarb 0.6.2 version. However, if update Scarb to the latest 0.7.0 version, several compilation errors occur. Since there is no mention anywhere with which version of Scarb the contracts are developed, it is recommended to support the latest available version of all tools.

Note: auditors may provide a patch by themselves, as we already checked the contracts with the latest Scarb version.

Recommendation:

Update the code to correspond to the latest Scarb version. Add a note in the repository regarding the version of tools the contracts were developed with.

Post-audti:

StarkDeFi team verified the decision to stick with the stable 0.6.2 version if Scarb. Though, the update with the next version will be included into the next protocol upgrade.

LOWEST-4	✔ Resolved

Usage of function not as a modifier.

StarkDRouter: _ensure().In the contract, the `_ensure`function is used as a regular function call within the body of other functions. While this does not pose a security risk, it does affect the readability of the code.

Recommendation:

Consider using a modifier-like pattern for the `_ensure`function to improve code readability.

Post-audit:

A Modifier-like pattern is used now.

LOWEST-5	✔ Resolved

Lack of Support for Tokens with Commissions.

The current implementation of the router contract does not support tokens that charge a commission on transfer. This could limit the types of tokens that can be used with this contract and could lead to incorrect calculations when such tokens are used.

Recommendation:

Consider adding support for tokens that charge a commission on transfer **OR** verify that support for these tokens was not specifically carried out.

Post-audit:

Support for tokens that charge a commission was added.

COMPLETE ANALYSIS (2ND ITERATION)

CRITICAL-1 ✔ Resolved

Incorrect calculation of f(x,y).

StarkDPair: function _f(), line 801.
In the _f function, the calculation of f(x,y) is performed using the equation f(x,y) = x * y * (x^2 + y^2). However, in the current implementation, the multiplication operation is performed as lhs * lhs instead of lhs * rhs. This mistake leads to an incorrect calculation of the function.

Recommendation:

Update the calculation of f(x,y) in the _f function to lhs * rhs.

Post-audit.
Calculation was updated.

CRITICAL-2 ✔ Resolved

Invalid array handling.

StarkDRouter:
swap_exact_tokens_for_tokens_supporting_fees_on_transfer_tokens(), line 219.
The issue arises when trying to access the last element of the `_path` array after a `pop_front()`operation, which results in a revert if the array contains only one element.

Recommendation:

A possible solution to this issue is adding a condition to check if the `_path`array is empty before accessing its last element. If the array is empty, use the `_route`variable (which was popped from `_path` earlier) as `_end_route`. If the array is empty, continue using the last element as `_end_route`.

Post-audit:

Condition was added.

HIGH-1	✔ Resolved

User will not get fees after withdraw liquidity.

Pair.cairo: burn(), claim_fees()
As from contract logic, users can collect fees if they provide liquidity. If the user removes liquidity without claiming fees, he will lose the fees he was collecting. Consider checking if the user has yet to collect fees before removing liquidity (burning tokens).

Recommendation:

Check fees for users before removing liquidity.

Post-audit.

Added claim_fees() in burn() function.

MEDIUM-1	✔ Resolved

Lack of validation in token transfers.

The contracts attempt to transfer tokens using the ERC20ABIDispatcher. This function only validates that the recipient and sender addresses are not zero. However, more is needed to ensure the transfer's success. For example, the recipient address could be a contract that does not accept token transfers or an address that was inputted incorrectly. Additionally, the function must check whether the contract has enough tokens to transfer, which could lead to failed transfers if the contract's balance is insufficient.

Recommendation:

Add additional validation checks.

Post-audit:

All accounts on Starknet are smart contracts. As such, there is no direct way to confirm a contract can accept a token without bloating the code. On the other hand, aside from ѕync`and ѕkim,` all other token transfers happening in the Pair`contract are meant to be low-level and must be after checks are done. The router handles this responsibility, which transfers from the user only after approval. If the user's balance is less than the approved amount, the Transfer will be aborted automatically. Likewise, for the pairFees,` an accurate record is kept for all tokens sent/withdrawn to/from the vault. During transfers, the records determine the exact amount to transfer out, ensuring transfers do work.

LOWEST-6	✔ Resolved

Usage of function not as a modifier.

StarkDFactory: assert_only_handler().In the contract, the `assert_only_handler`function is used as a regular function call within the body of other functions. While this does not pose a security risk, it does affect the readability of the code.

Recommendation:

Consider using a modifier-like pattern for the `assert_only_handler` function to improve code readability.

Post-audit:

A Modifier-like pattern is used now.

LOWEST-7	✔ Resolved

Confusing contract name.

pairFee.cairo.
When looking at the Pair contract, it needs a vault to set up the contract correctly. It could be confusing in this case as no vault contract or anything connected to the vault cannot be found. For better readability, a contract with a `vault`in its name should be written as it is not clear at first glance that pairFees.cairo contract is the vault contract that needs to be provided for pair set up.

Recommendation:

Change pairFee contract name to feeVault or any name with `Vault` in it.

Post-audit:

Name was changed.

LOWEST-8	✔ Resolved

Lack of validation.

1) Pair.cairo: mint(), line 325.
The mint function does not perform proper input validation for the `to`and token addresses. It assumes that the `to` address is not equal to `token0`and `token1`, which can lead to unexpected behavior if the to address is one of the tokens in the pair.
2) router.cairo: swap_exact_tokens_for_tokens, swap_exact_tokens_for_tokens_supporting_fees_on_transfer_tokens
The functions described below do not check if the input amount is zero and that can lead to unexpected behavior.

Recommendation:

Add input validation.

Post-audit:

From the client: `mint()`in Pair.cairo inherits `_mint`from the ERC20 token which checks that `to`is not zero address;
`get_amount_out`in Pair.cairo has validation for zero amounts.

LOWEST-9	✔ Resolved

Lack of breaker mechanism.

The contracts do not have any circuit breaker mechanism in place to pause or stop certain operations in case of emergencies or unexpected issues.

Recommendation:

Implement a circuit breaker mechanism to allow for the pausing or stopping of certain operations in case of emergencies or unexpected issues.

Post-audit:

Added pause mechanism to contracts.

LOWEST-10	✔ Resolved

Lack of Validation for Token Swap Path.

router.cairo: swap_exact_tokens_for_tokens, swap_exact_tokens_for_tokens_supporting_fees_on_transfer_tokens.

The `swap_exact_tokens_for_tokens` function in the router.cairo contract currently lacks validation for the provided token swap path. The function is expected to handle a sequence of token swaps from one token to another, following a specified path. However, there is no check in place to ensure that the path is valid. A valid path should have the output token of one swap as the input token for the next swap. For example, if the first swap is from token 0 to token 1, the next swap should be from token 1 to token 2, and so on.

Without this validation, the function could potentially execute a sequence of swaps that do not follow a logical path, leading to unexpected results and potential loss of funds for the user.

Recommendation:

Add a validation step to check that each output token in the path matches the input token of the next swap.

Post-audit:

Validation was added.

STANDARD CHECKLIST

		factory.cairo pair.cairo pairFees.cairo	router.cairo utils.cairo
✓	L1-L2 Addresses Conversion	Pass	Pass
✓	Access Management Hierarchy	Fail	Pass
✓	Integer Division and Overflows	Pass	Pass
✓	Unexpected Tokens / Dust Attack	Pass	Pass
✓	Public Interface Constrains	Pass	Pass
✓	Hidden Malicious Code	Pass	Pass
✓	Entropy Illusion (Lack of Randomness)	Pass	Pass
✓	External Contract Referencing	Pass	Pass
✓	Incorrect Parameters	Pass	Pass
✓	Unchecked CALL Return Values	Pass	Pass
✓	Tx Order Dependency	Pass	Pass
✓	General Denial Of Service (DOS)	Pass	Pass
✓	View State Modifications	Pass	Pass
✓	Floating Points and Precision	Pass	Pass
✓	Namespace Storage Var Collision	Pass	Pass
✓	Signatures Replay	Pass	Pass
✓	Pool Asset Security (backdoors in the underlying tokens)	Fail	Pass

Note: standard checklist is only a part of performed checks, which reflects review against common mistakes and vulnerabilities.

CODE COVERAGE AND TEST RESULTS FOR ALL FILES, PREPARED BY BLAIZE SECURITY TEAM (1ST ITERATION)

```
test starkDefi::tests::pair::test_pair::test_k ... ok

test starkDefi::tests::audit::audit_tests::test_change_fee_receiver ... ok
test starkDefi::tests::audit::audit_tests::test_create_pair ... ok
test starkDefi::tests::audit::audit_tests::test_create_same_pair ... ok
test starkDefi::tests::audit::audit_tests::test_swap_multiple_times ... ok
test starkDefi::tests::audit::audit_tests::test_add_liquidity ... ok
test starkDefi::tests::audit::audit_tests::test_swap_tokens ... ok
test starkDefi::tests::audit::audit_tests::test_swap_zero ... ok
test starkDefi::tests::audit::audit_tests::test_remove_zero_liquidity ... ok
test starkDefi::tests::audit::audit_tests::test_change_fee_setter ... ok
test starkDefi::tests::audit::audit_tests::test_remove_liquidity ... ok
test starkDefi::tests::audit::audit_tests::test_create_pair_zero_address ... ok
test starkDefi::tests::audit::audit_tests::test_check_received_fee ... ok
test starkDefi::tests::audit::audit_tests::test_create_same_token_pair ... ok
test starkDefi::tests::audit::audit_tests::test_slipage ... ok
test starkDefi::tests::audit::audit_tests::test_deploy_router_invalid ... ok
test starkDefi::tests::audit::audit_tests::test_deploy_factory_invalid ... ok
test starkDefi::tests::audit::audit_tests::test_create_pair_token_identical ...
ok
test starkDefi::tests::audit::audit_tests::test_create_pair_token_zero ... ok
test starkDefi::tests::audit::audit_tests::test_swap_large_amount ... ok
test starkDefi::tests::audit::audit_tests::test_multiple_users ... ok
```

During testing, auditors have checked several critical components of the DEX including:

- Creation of liquidity pairs;
- Adding and removing of liquidity;
- Exchanging of tokens and changing of reserves in pair;
- Changing of cumulative prices;
- Tests of slippage and flashloans;
- Tests of the correctness of the main AMM invariant.

Tests were conducted with different values of tokens including small dusting amounts and large amounts close to the maximum possible. All the tests have passed correctly verifying the correctness of DEX implementation.

CODE COVERAGE AND TEST RESULTS FOR ALL FILES, PREPARED BY BLAIZE SECURITY TEAM (2ND ITERATION)

```
test starkDefi::tests::audit::audit_tests::test_set_fees_not_allowed ... ok
test starkDefi::tests::audit::audit_tests::test_set_fees_invalid ... ok
test starkDefi::tests::audit::audit_tests::test_set_fee ... ok
test starkDefi::tests::audit::audit_tests::test_create_pair_token_zero ... ok
test starkDefi::tests::audit::audit_tests::test_set_custom_pair_fee ... ok
test starkDefi::tests::audit::audit_tests::test_change_fee_receiver ... ok
test starkDefi::tests::audit::audit_tests::test_set_fee_handler ... ok
test starkDefi::tests::audit::audit_tests::test_create_pair_token_identical ...
ok
test starkDefi::tests::audit::audit_tests::test_create_pair ... ok
test starkDefi::tests::audit::audit_tests::test_remove_zero_liquidity ... ok
test starkDefi::tests::audit::audit_tests::test_create_stable_pair ... ok
test starkDefi::tests::audit::audit_tests::test_deploy_factory_invalid ... ok
test starkDefi::tests::audit::audit_tests::test_slipage ... ok
test starkDefi::tests::audit::audit_tests::test_deploy_router_invalid ... ok
test starkDefi::tests::audit::audit_tests::test_swap_tokens ... ok
test starkDefi::tests::audit::audit_tests::test_remove_liquidity ... ok
test starkDefi::tests::audit::audit_tests::test_add_liquidity ... ok
test starkDefi::tests::audit::audit_tests::test_swap_zero ... ok
test starkDefi::tests::audit::audit_tests::test_swap_large_amount ... ok
test starkDefi::tests::audit::audit_tests::test_swap_multiple_times ... ok
test starkDefi::tests::audit::audit_tests::test_swap_tokens_invalid_path ... ok
test starkDefi::tests::audit::audit_tests::test_k_stable ... ok
test starkDefi::tests::audit::audit_tests::test_k ... ok
test starkDefi::tests::audit::audit_tests::test_swap_with_fee_regular_token ...
ok
test starkDefi::tests::audit::audit_tests::test_check_received_fee ... ok
test starkDefi::tests::audit::audit_tests::test_swap_with_fee ... ok
test starkDefi::tests::audit::audit_tests::test_k_stable_fee ... ok
test starkDefi::tests::audit::audit_tests::test_pause_contract ... ok
test starkDefi::tests::audit::audit_tests::test_pair_fee ... ok
test result: ok. 29 passed; 0 failed; 0 ignored; 0 filtered out;
```

CODE COVERAGE AND TEST RESULTS FOR ALL FILES, PREPARED BY STARKDEFI TEAM (1ST ITERATION)

test starkDefi::tests::factory::test_factory::test_set_fee_to_setter_new_setter ... ok
test starkDefi::tests::factory::test_factory::test_constructor ... ok
test starkDefi::tests::factory::test_factory::test_set_fee_to_setter_not_allowed ... ok
test starkDefi::tests::factory::test_factory::test_set_fee_to_setter_invalid ... ok
test starkDefi::tests::factory::test_factory::test_sort_tokens ... ok
test starkDefi::tests::factory::test_factory::test_sort_tokens_identical ... ok
test starkDefi::tests::factory::test_factory::test_sort_tokens_invalid_token0 ... ok
test starkDefi::tests::pair::test_pair::test_constructor ... ok
test starkDefi::tests::factory::test_factory::test_deployed_factory ... ok
test starkDefi::tests::factory::test_factory::test_fee_to ... ok
test starkDefi::tests::factory::test_factory::test_fee_to_setter ... ok
test starkDefi::tests::pair::test_pair::test_mint_no_zero_tokens ... ok
test starkDefi::tests::factory::test_factory::test_class_hash_for_pair_contract ... ok
test starkDefi::tests::factory::test_factory::test_all_pairs_length ... ok
test starkDefi::tests::factory::test_factory::test_get_pair ... ok
test starkDefi::tests::factory::test_factory::test_get_pair_not_found ... ok
test starkDefi::tests::factory::test_factory::test_all_pairs ... ok
test starkDefi::tests::factory::test_factory::test_create_pair ... ok
test starkDefi::tests::pair::test_pair::test_swap_insufficient_liquidity ... ok
test starkDefi::tests::factory::test_factory::test_deployed_create_pair ... ok
test starkDefi::tests::factory::test_factory::test_create_pair_twice ... ok
test starkDefi::tests::factory::test_factory::test_create_pair_invalid_token ... ok
test
starkDefi::tests::router::test_router::test_router_get_amount_out_insufficient_amo
unt ... ok
test starkDefi::tests::pair::test_pair::test_burn_remove_all_liquidity ... ok
test starkDefi::tests::factory::test_factory::test_create_pair_identical_token ... ok
test starkDefi::tests::pair::test_pair::test_ming_not_enough_tokens ... ok
test starkDefi::tests::factory::test_factory::test_create_pair_pair_exists ... ok
test starkDefi::tests::factory::test_factory::test_set_fee_to ... ok
test starkDefi::tests::pair::test_pair::test_deployed_pair ... ok
test starkDefi::tests::factory::test_factory::test_set_fee_to_not_allowed ... ok
test starkDefi::tests::factory::test_factory::test_set_fee_to_setter ... ok
test starkDefi::tests::pair::test_pair::test_swap_invalid_to ... ok

```
test starkDefi::tests::router::test_router::test_router_add_liquidity_insufficient_b ...
ok
test starkDefi::tests::pair::test_pair::test_burn_insufficient_liquidity ... ok
test starkDefi::tests::router::test_router::test_router_remove_liqudity_less_B ... ok
test
starkDefi::tests::router::test_router::test_router_get_amount_out_insufficient_liqui
dity ... ok
test starkDefi::tests::pair::test_pair::test_swap_token0_for_token1 ... ok
test starkDefi::tests::pair::test_pair::test_swap_token1_for_token0 ... ok
test starkDefi::tests::pair::test_pair::test_mint_more_lp ... ok
test starkDefi::tests::pair::test_pair::test_swap_insufficient_input_amount ... ok
test starkDefi::tests::router::test_router::test_router_get_amount_in ... ok
test starkDefi::tests::router::test_router::test_router_quote ... ok
test starkDefi::tests::pair::test_pair::test_swap_insufficient_output_amount ... ok
test starkDefi::tests::pair::test_pair::test_skim ... ok
test starkDefi::tests::pair::test_pair::test_swap_invariant_k ... ok
test starkDefi::tests::router::test_router::test_router_add_liquidity_insufficient_a ...
ok
test starkDefi::tests::router::test_router::test_deploy_router ... ok
test starkDefi::tests::router::test_router::test_router_quote_insufficient_amount ...
ok
test starkDefi::tests::pair::test_pair::test_mint ... ok
test starkDefi::tests::pair::test_pair::test_sync ... ok
test
starkDefi::tests::router::test_router::test_router_get_amount_in_insufficient_amo
unt ... ok
test starkDefi::tests::pair::test_pair::test_fee_on ... ok
test starkDefi::tests::router::test_router::test_router_add_new_liquidity ... ok
test starkDefi::tests::router::test_router::test_router_quote_insufficient_liquidity ...
ok
test starkDefi::tests::pair::test_pair::test_burn ... ok
test starkDefi::tests::router::test_router::test_router_remove_all_liquidity ... ok
test starkDefi::tests::router::test_router::test_router_get_amount_in_insufficient
_liquidity ... ok
test starkDefi::tests::router::test_router::test_router_remove_liqudity_some ... ok
test starkDefi::tests::router::test_router::test_router_add_more_liquidity ... ok
test starkDefi::tests::router::test_router::test_router_remove_liqudity_less_A ... ok
```

```
test starkDefi::tests::router::test_router::test_router_get_amount_out ... ok
test starkDefi::tests::router::test_router::test_router_get_amounts_out ... ok
test starkDefi::tests::router::test_router::test_router_swap_tokens_for_exact
_tokens_2 ... ok
test starkDefi::tests::router::test_router::test_router_swap_tokens_for_exact
_tokens ... ok
test starkDefi::tests::router::test_router::test_router_swap_exact_tokens_for
_tokens ... ok
test starkDefi::tests::router::test_router::test_router_swap_exact_tokens_for
_tokens_invalid ... ok
test starkDefi::tests::router::test_router::test_router_swap_tokens_for_exact
_tokens_3 ... ok
test starkDefi::tests::router::test_router::test_router_swap_exact_tokens_for
_tokens_2 ... ok
test starkDefi::tests::router::test_router::test_router_get_amounts_out_invalid
_path ... ok
test starkDefi::tests::router::test_router::test_router_swap_exact_tokens_for
_tokens_multiple ... ok
test starkDefi::tests::router::test_router::test_router_get_amounts_in ... ok
test starkDefi::tests::router::test_router::test_router_swap_exact_tokens_for
_tokens_3 ... ok
test starkDefi::tests::router::test_router::test_router_get_amounts_in_invalid_path
... ok
test starkDefi::tests::router::test_router::test_router_swap_tokens_for_exact
_tokens_invalid ... ok
test starkDefi::tests::router::test_router::test_router_swap_tokens_for_token
_multiply ... ok
test result: ok. 75 passed; 0 failed; 0 ignored; 0 filtered out;
```

CODE COVERAGE AND TEST RESULTS FOR ALL FILES, PREPARED BY STARKDEFI TEAM (2ND ITERATION)

test starkDefi::tests::factory::test_factory::test_constructor ... ok
test starkDefi::tests::factory::test_factory::test_deployed_factory ... ok
test starkDefi::tests::factory::test_factory::test_fee_to ... ok
test starkDefi::tests::factory::test_factory::test_get_fees ... ok
test starkDefi::tests::factory::test_factory::test_fee_handler ... ok
test starkDefi::tests::factory::test_factory::test_set_custom_pair_fee ... ok
test starkDefi::tests::factory::test_factory::test_class_hash_for_pair_contract ... ok
test starkDefi::tests::factory::test_factory::test_all_pairs_length ... ok
test starkDefi::tests::factory::test_factory::test_set_custom_pair_fee_not_allowed ...
test starkDefi::tests::factory::test_factory::test_get_pair ... ok
test starkDefi::tests::pair::test_volatile_pair::test_vPair_mint ... ok
test starkDefi::tests::factory::test_factory::test_get_pair_not_found ... ok
test starkDefi::tests::factory::test_factory::test_all_pairs ... ok
test starkDefi::tests::factory::test_factory::test_set_custom_pair_fee_too_high ... ok
test starkDefi::tests::factory::test_factory::test_set_custom_pair_fee_invalid_pair ... (
test starkDefi::tests::factory::test_factory::test_set_fee_handler ... ok
test starkDefi::tests::factory::test_factory::test_set_fee_handler_new_handler ... ok
test starkDefi::tests::factory::test_factory::test_create_pair ... ok
test starkDefi::tests::factory::test_factory::test_set_fee_handler_not_allowed ... ok
test starkDefi::tests::router::test_router::test_deploy_router ... ok
test starkDefi::tests::factory::test_factory::test_set_fee_handler_invalid ... oktest
starkDefi::tests::factory::test_factory::test_sort_tokens ... ok
test starkDefi::tests::factory::test_factory::test_sort_tokens_identical ... ok
test starkDefi::tests::factory::test_factory::test_sort_tokens_invalid_token0 ... ok
test starkDefi::tests::factory::test_factory::test_deployed_create_pair ... ok
test starkDefi::tests::pair::test_pair_shared::test_pair_constructor ... ok
test starkDefi::tests::factory::test_factory::test_create_pair_twice ... ok
test starkDefi::tests::factory::test_factory::test_create_pair_invalid_token ...
ok
test starkDefi::tests::factory::test_factory::test_create_pair_identical_token ... ok
test starkDefi::tests::factory::test_factory::test_create_pair_pair_exists ... oktest
starkDefi::tests::factory::test_factory::test_set_fee_to ... ok
test starkDefi::tests::factory::test_factory::test_set_fee_to_not_allowed ... ok
test starkDefi::tests::factory::test_factory::test_set_fees ... ok
test starkDefi::tests::factory::test_factory::test_set_fees_not_allowed ... ok

```
test starkDefi::tests::router::test_router::test_router_add_new_liquidity ... ok
test starkDefi::tests::factory::test_factory::test_set_fees_invalid ... ok
test starkDefi::tests::factory::test_factory::test_set_fees_max ... ok
test starkDefi::tests::pair::test_pair_shared::test_deployed_pair ... ok
test starkDefi::tests::pair::test_pair_shared::test_mint_no_zero_tokens ... ok
test starkDefi::tests::pair::test_pair_shared::test_fees_collected_on_swap ... oktest
starkDefi::tests::pair::test_stable_pair::test_sPair_get_amount_out ... ok
test starkDefi::tests::pair::test_pair_shared::test_mint_not_enough_tokens ... oktest
starkDefi::tests::pair::test_pair_shared::test_get_amount_out_insufficient_in ... ok
test starkDefi::tests::router::test_router::test_router_add_more_liquidity ... oktest
starkDefi::tests::pair::test_stable_pair::test_sPair_swap_invariant_k ... oktest
starkDefi::tests::pair::test_pair_shared::test_swap_insufficient_liquidity ... ok
test starkDefi::tests::pair::test_pair_shared::test_mint ... ok
test starkDefi::tests::pair::test_stable_pair::test_sPair_mint ... ok
test
starkDefi::tests::pair::test_pair_shared::test_vPair_swap_insufficient_output_amo
unt ... ok
test starkDefi::tests::router::test_router::test_router_add_liquidity_insufficient_b ...
ok
test starkDefi::tests::pair::test_pair_shared::test_skim ... ok
test starkDefi::tests::pair::test_stable_pair::test_sPair_mint_unmatched ... ok
test starkDefi::tests::pair::test_volatile_pair::test_vPair_constructor ... ok
test starkDefi::tests::pair::test_pair_shared::test_swap_invalid_to ... ok
test starkDefi::tests::pair::test_volatile_pair::test_deployed_vPair ... ok
test starkDefi::tests::pair::test_pair_shared::test_sync ... ok
test starkDefi::tests::pair::test_pair_shared::test_swap_insufficient_input_amount
... ok
test starkDefi::tests::router::test_router::test_router_add_liquidity_insufficient_a ...
ok
test starkDefi::tests::pair::test_volatile_pair::test_vPair_mint_more_lp ... ok
test starkDefi::tests::pair::test_pair_shared::test_burn ... ok
test starkDefi::tests::router::test_router::test_router_remove_all_liquidity ... ok
test starkDefi::tests::pair::test_volatile_pair::test_vPair_swap_token0_for_token1 ...
ok
test starkDefi::tests::pair::test_pair_shared::test_claim_fees ... ok
test starkDefi::tests::pair::test_stable_pair::test_sPair_constructor ... ok
test starkDefi::tests::pair::test_pair_shared::test_burn_remove_all_liquidity ... ok
test starkDefi::tests::pair::test_stable_pair::test_deployed_sPair ... ok
```

```
test starkDefi::tests::pair::test_volatile_pair::test_vPair_swap_token1_for_token0 ...
ok
test starkDefi::tests::router::test_router::test_router_remove_liqudity_some ... ok
test starkDefi::tests::pair::test_pair_shared::test_burn_insufficient_liquidity ... ok
test starkDefi::tests::pair::test_volatile_pair::test_vPair_get_amount_out ... ok
test starkDefi::tests::pair::test_volatile_pair::test_vPair_swap_invariant_k ... ok
test starkDefi::tests::pair::test_stable_pair::test_sPair_mint_more_lp ... ok
test starkDefi::tests::router::test_router::test_router_remove_liqudity_less_A ... ok
test starkDefi::tests::router::test_router::test_router_quote_insufficient_amount ...
ok
test starkDefi::tests::pair::test_stable_pair::test_sPair_swap_token0_for_token1 ...
ok
test starkDefi::tests::router::test_router::test_router_remove_liqudity_less_B ... ok
test starkDefi::tests::router::test_router::test_router_quote_insufficient_liquidity ...
ok
test starkDefi::tests::pair::test_stable_pair::test_sPair_swap_token1_for_token0 ...
ok
test starkDefi::tests::router::test_router::test_router_quote ... ok
test starkDefi::tests::router::test_router::test_router_get_amount_out ... ok
teststarkDefi::tests::router::test_router::test_router_get_amounts_out_invalid_path
... ok
teststarkDefi::tests::router::test_router::test_router_get_amount_out_insufficient_a
mount ... ok
teststarkDefi::tests::router::test_router::test_router_get_amount_out_insufficient_li
quidity ... ok
teststarkDefi::tests::router::test_router::test_router_swap_exact_tokens_for_token
s_2 ... ok
test starkDefi::tests::router::test_router::test_router_get_amounts_out ... ok
teststarkDefi::tests::router::test_router::test_router_swap_exact_tokens_for_token
s_3 ... ok
teststarkDefi::tests::router::test_router::test_router_swap_exact_tokens_for_tokens
... ok
teststarkDefi::tests::router::test_router::test_router_swap_exact_tokens_for_tokens
_invalid ... ok
teststarkDefi::tests::router::test_router::test_router_swap_exact_tokens_for_tokens
_multiple ... ok
```

```
test starkDefi::tests::router::test_router::test_router_swap_exact_tokens_for
_tokens_invalid_path_progression ... ok
test starkDefi::tests::router::test_router::test_router_swap_exact_tokens_for_
tokens_supporting_fees_on_transfer_tokens_invalid_path_progression ... ok
test starkDefi::tests::router::test_router::test_router_swap_exact_tokens_for_
tokens_supporting_fees_on_transfer_tokens_multiple ... ok
test result: ok. 92 passed; 0 failed; 0 ignored; 0 filtered out;
```

DISCLAIMER

The information presented in this report is an intellectual property of the customer, including all the presented documentation, code databases, labels, titles, ways of usage, as well as the information about potential vulnerabilities and methods of their exploitation. This audit report does not give any warranties on the absolute security of the code. Blaize.Security is not responsible for how you use this product and does not constitute any investment advice.

Blaize.Security does not provide any warranty that the working product will be compatible with any software, system, protocol or service and operate without interruption. We do not claim the investigated product is able to meet your or anyone else's requirements and be fully secure, complete, accurate, and free of any errors and code inconsistency.

We are not responsible for all subsequent changes, deletions, and relocations of the code within the contracts that are the subjects of this report.

You should perceive Blaize.Security as a tool, which helps to investigate and detect the weaknesses and vulnerable parts that may accelerate the technology improvements and faster error elimination.