Blaize.Security

February 16th, 2024 / V. 1.0

StarkDefi.

STARKDEFI

SMART CONTRACT AUDIT

TABLE OF CONTENTS

AUDIT RATING

SCORE **9.85**/10

The scope of the project includes StarkDefi:

locker.cairo

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

The source code of the smart contract was taken from the branch: feat/locker.

Initial commit:

- efc7fb1dc72bc7c458d9b7b59753ac3688a34819

During the audit process, code fixes occurred in the branch: fix/audit.

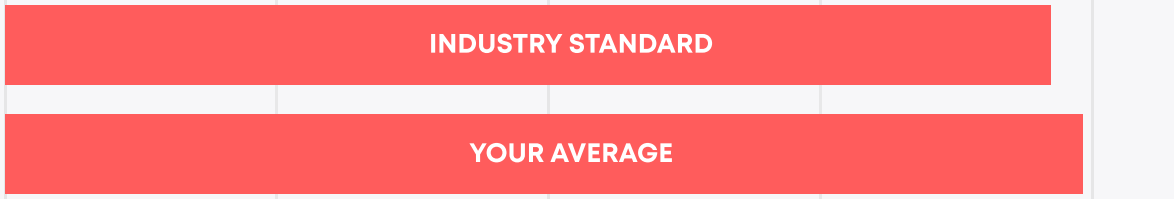
Final commit:

- 652eddd1be9bb3e6344ead951c809368656770f9

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 between **January 29th, 2024** and **February 16th, 2024.**

Testable code



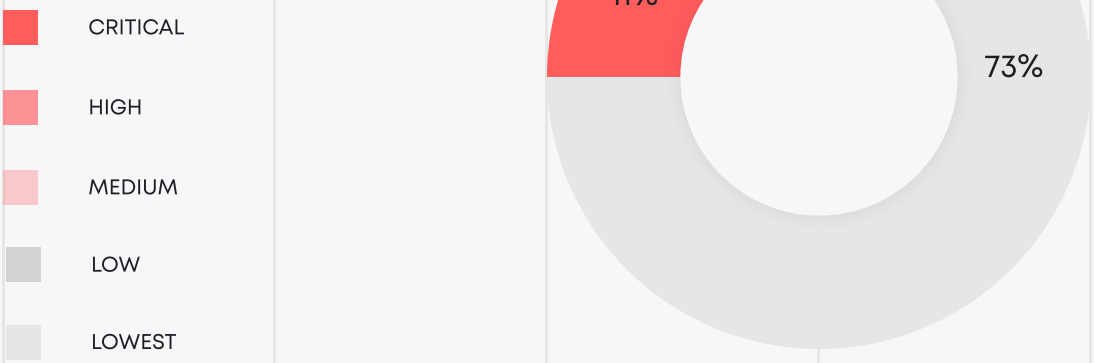
Auditors approved code as testable within the industry standard.

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 19 problems were found. 18 issues were fixed or verified by the StarkDefi team.

	FOUND	FIXED/VERIFIED
Critical	2	2
High	0	0
Medium	2	2
Low	1	1
Lowest	14	13

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 conducted the audit of the StarkDefi Locker contract, a key part of the StarkDefi project offering DeFi services on StarkNet. The Cairo contract (for the StarkNet chain) enables user-initiated token locks for both ERC20 and ERC721 tokens and features a fixed fee collection system. Its design includes an upgradeability function that ensures future adaptability and mechanisms to retrieve tokens sent to the contract by accident. Also, the contract integrates OpenZeppelin's secure and tested contracts for Cairo.

Blaize Security conducted comprehensive testing during the audit to verify the contract's key functionalities, including token locking and fee calculation mechanisms. The team assessed procedures for locking and unlocking tokens, the accuracy of fee computations, and the effectiveness of the contract's upgrade functionality. Blaize team prepared its own set of tests for complex scenarios, placing focus on the "Medium-1" issue related to the absence of user lock limits. Despite StarkDefi's choice to allow unlimited locks for users, which aims to keep the platform flexible and user-friendly, Blaize Security noted the importance of mitigating potential risks through evaluation and considering options for optimization. Auditors still have concerns regarding this part of the functionality. Thus, the protocol failed the DDoS check since the current way of lock handling introduces such a risk. Also, auditors need to mention that the contract is upgradeable. Thus, the protocol fails the check for backdoors - as the upgradeability is controllable but still a backdoor. Auditors highly recommend a secured flow for key management and upgrade process and monitoring regarding the risk of locks blocking the storage.

From other points of view, the contract adheres to high-security standards and follows best practices for writing smart contracts for the StarkNet ecosystem. It is well-documented, facilitating understanding and interaction with the contract's features. However, auditors need to mention that no native tests were presented for the Locker functionality. Nevertheless, StarkDefi's proactive stance on addressing audit findings underscores their commitment to providing secure and operational DeFi services. This approach solidifies user confidence and underlines the continuous evolution and maturity of the protocol.

	RATING
Security	9.5
Logic optimization	9.7
Code quality	9.9
Testable code*	9.8
Total	9.7

*Testable code mark was calculated only according to Blaze Security tests.

PROTOCOL OVERVIEW

DESCRIPTION

The StarkDefi Locker Contract is a Cairo-based smart contract designed for the StarkNet ecosystem. It provides a time-based locking mechanism for ERC20 and ERC721 (NFT) tokens. The primary functionality of the contract allows users to deposit tokens into a lock that will only allow withdrawal after a predefined period.

FEE SYSTEM

The fee system in the StarkDefi Locker contract represents a mechanism that charges a fixed fee for utilizing the token locking function.

Features of the fee system:

- Fixed Price: The fee for locking tokens is set as a fixed amount and does not vary with the size of the lock or any other factors.

- Administrative Configuration: Fees are configurable by the contract administrator, who can modify the fee amount through dedicated functions.

- Fee Recipient Address: The contract allows the administrator to set an address where the collected fees are directed.

- Fee Disabling Option: The system offers the ability to disable fee collection.

ROLES AND RESPONSIBILITIES

1. Owner (OWNER_ROLE):
- The high-level administration and management of the contract.
- Can initiate upgrades of the contract.
- Has authority over role assignments and revocations.

2. Admin (ADMIN_ROLE):
- Manages contract parameters like fees and feature toggles.
- Controls adding or removing addresses from whitelists.
- May adjust feature settings like enabling or disabling NFT locks.

3. Pauser (PAUSER_ROLE):
- Holds the authority to pause contract functionality, preventing any further state-altering transactions.

4. Unpauser (UNPAUSER_ROLE):
- Authorized to unpause the contract, restoring its operational capabilities after the pause conditions have been resolved.

5. Fee Receiver (FEE_RECEIVER):
- The intended recipient of the fees collected for token locking services.

6. User (General Users):
- Any individual or entity interacting with the smart contract who does not hold a specialized role is considered a general user.
- Users can create locks for ERC20 and ERC721 tokens.
- They can also manage their locks by performing actions like withdrawals (post-unlock), extending the lock duration, transferring the lock to another address, or splitting the lock.

SETTINGS

1. Lock Fee Settings:
   - These include the amount of fee charged for locking tokens, the token in which fees are paid, and the recipient address for these fees.
   - Incorrect settings could lead to prohibitively high fees discouraging users from using the service, or too low fees that might not cover the operational costs or expected revenue. Setting an incorrect recipient could also mean that fees are misdirected to the wrong address.

2. Roles Assignment:
   - Defining which addresses have OWNER_ROLE, ADMIN_ROLE, PAUSER_ROLE, and UNPAUSER_ROLE.
   - Misassignment could result in unauthorized access to critical functions such as pausing the contract or upgrading it, potentially leading to pauses at inopportune times, unauthorized upgrades and so on.

3. NFT Locking Capability:
   - A toggle setting that enables or disables the locking of NFTs within the contract.
   - If incorrectly set to disable NFT locks when they should be available, the contract would fail to serve a segment of its user base, limiting its functionality.

4. Protocol Pause and Unpause Functionality:
   - Protocols for pausing and unpausing the contract, typically used in emergencies or during upgrades.
   - An unintended pause could freeze all contract operations, leading to user frustration and potential financial loss. Also, not being able to pause the contract quickly in response to a security threat could lead to exploitation.

LIST OF VALUABLE ASSETS

1. Locked Tokens (ERC20 and ERC721):
The main assets managed by the contract are the ERC20 tokens and ERC721 NFTs that users lock within the system.
2. Fees Collected:
The fees that are configured and collected for token-locking services generate value for the contract owners or designated fee receivers. They serve as a revenue stream and are considered financial assets.

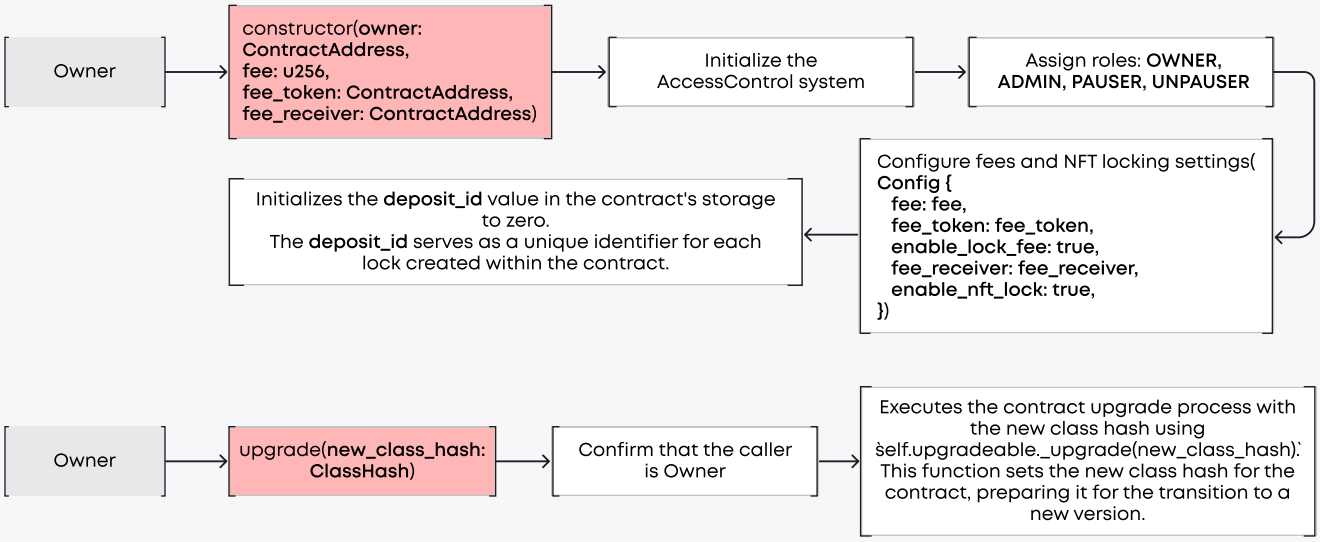
The contract features rescue functions that allow the withdrawal of unallocated and accidentally sent assets. "Unallocated" refers to assets not involved in any user-initiated lock process; for example, if an ERC20 token 'A' is already locked by users within the contract, it cannot be rescued until the rightful owner withdraws the lock. However, if an ERC20 token 'B' is sent to the contract by mistake and has not been locked, it can be withdrawn through the `rescue_tokens` function. Similarly, the `rescue_nft` function allows the retrieval of accidentally sent NFTs unless they are part of an existing lock.

DEPLOYMENT SCRIPT

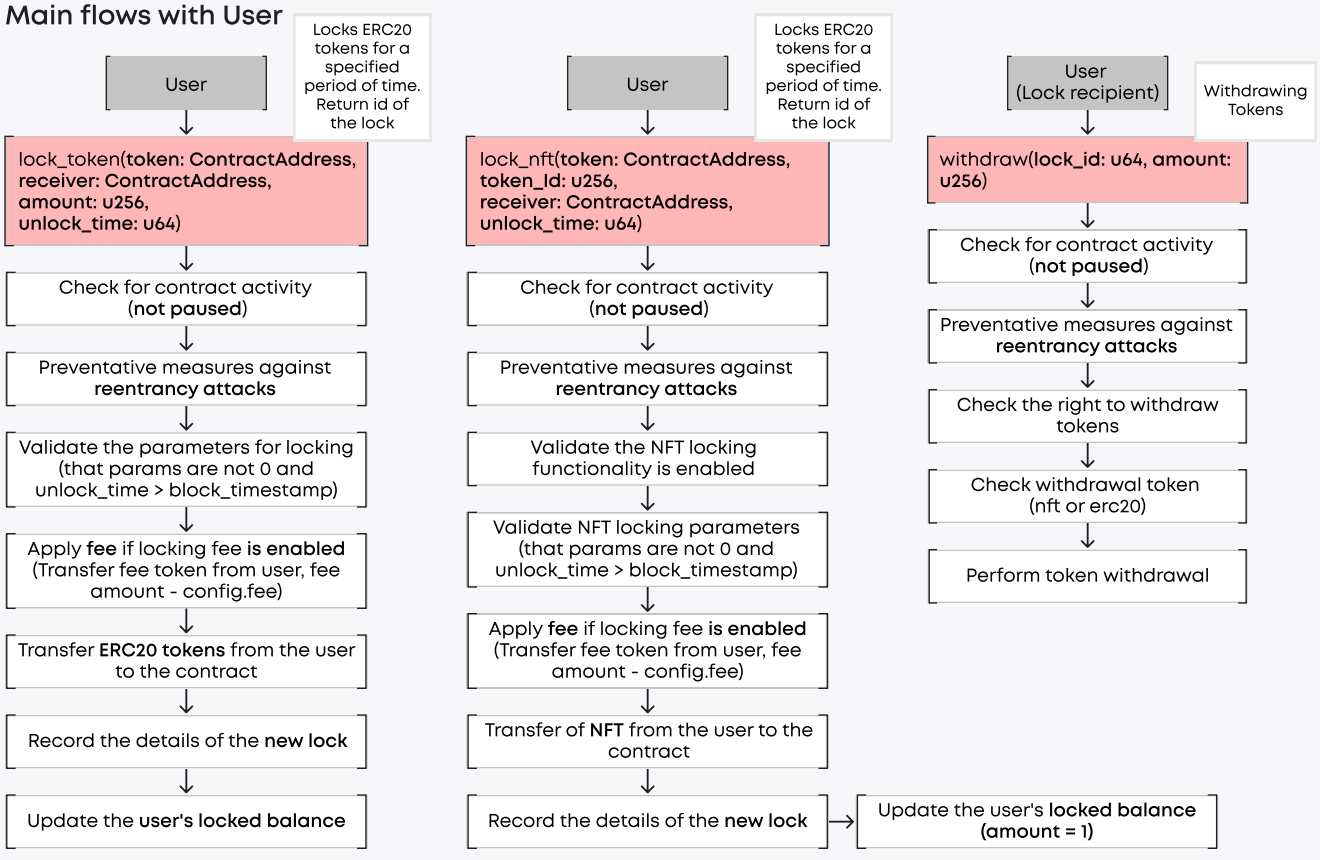
As for StarkNet, contracts deployed using CLI commands. StarkDefi team is using sncast CLI tool to deploy contracts. More information can be found on https://foundry-rs.github.io/starknet-foundry/starknet/index.html.

STARKDEFI (LOCKER)

Initialization with Owner

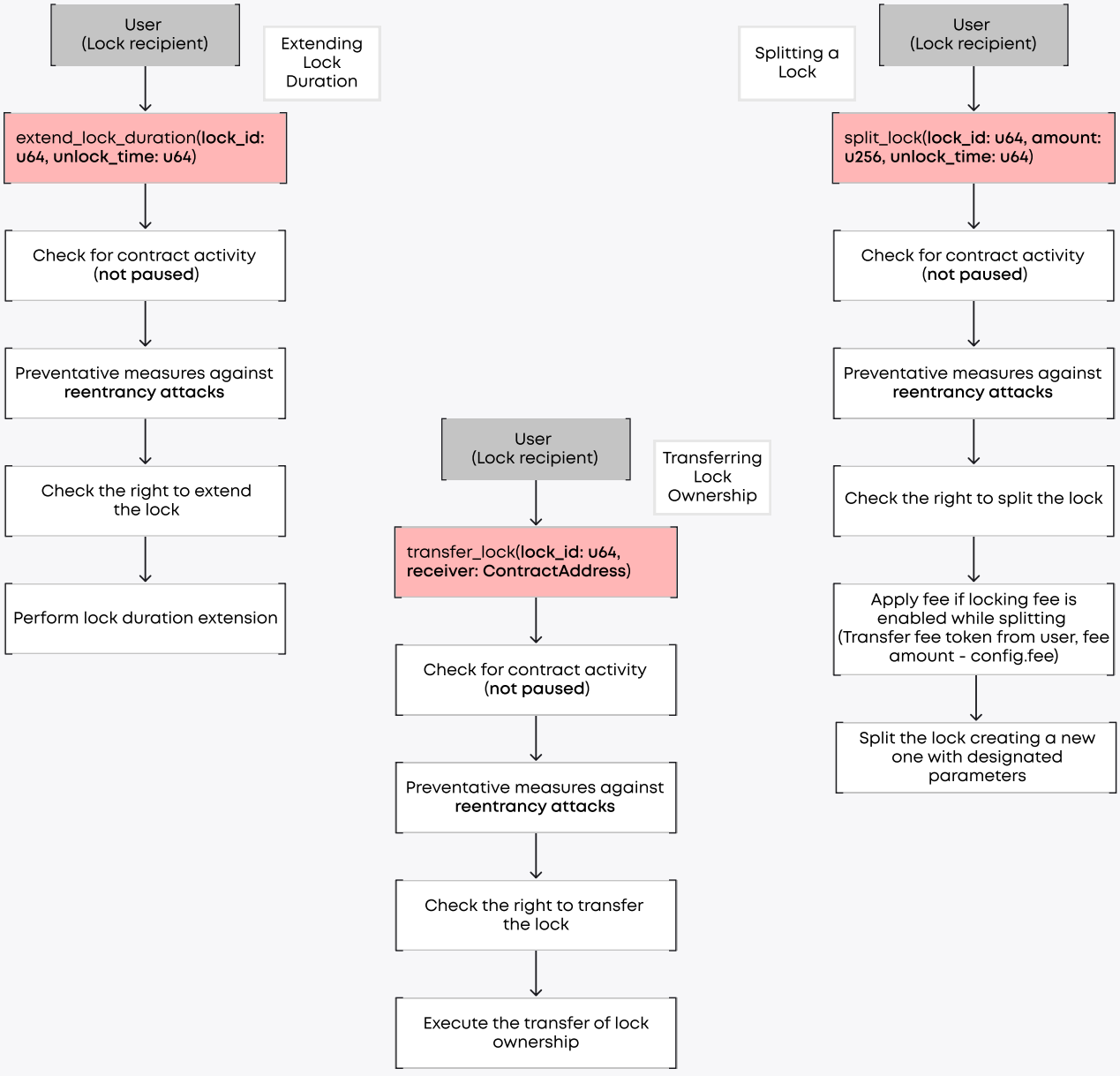


Main flows with User



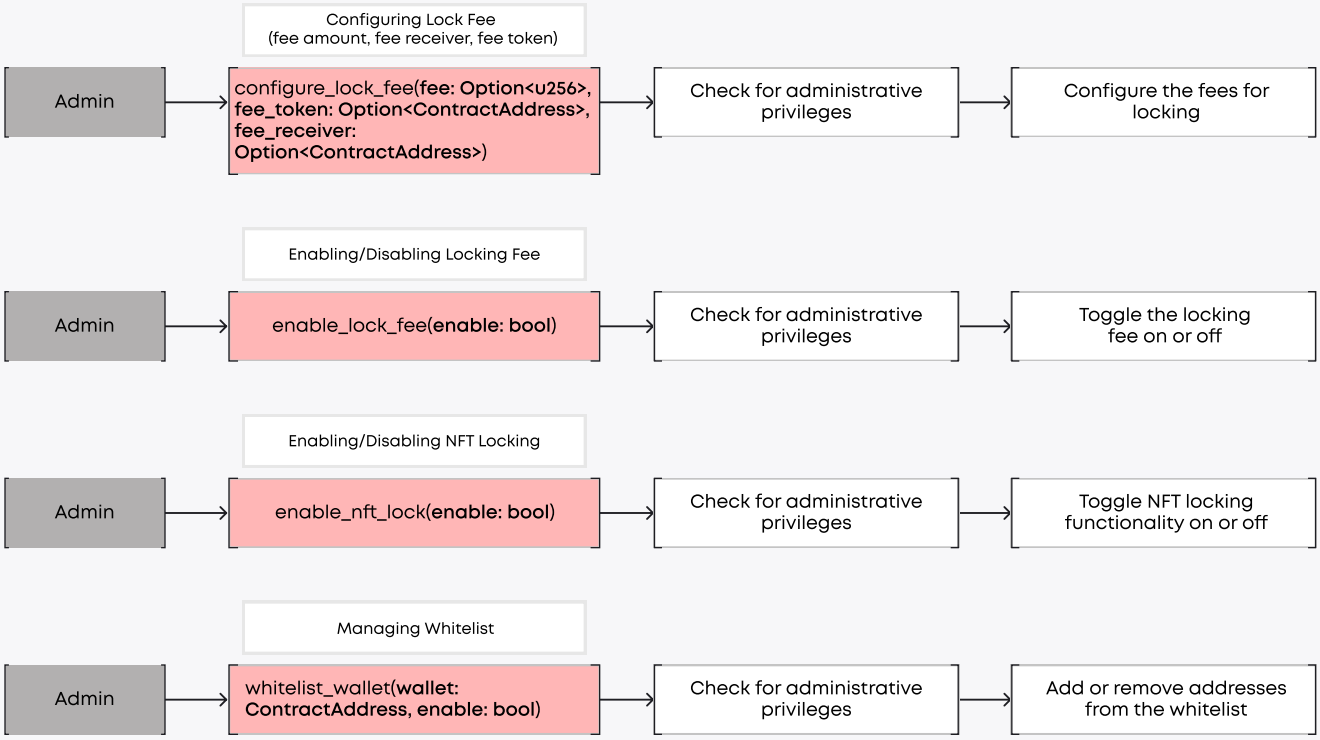
STARKDEFI (LOCKER)

Settings with User (as Lock recipient)

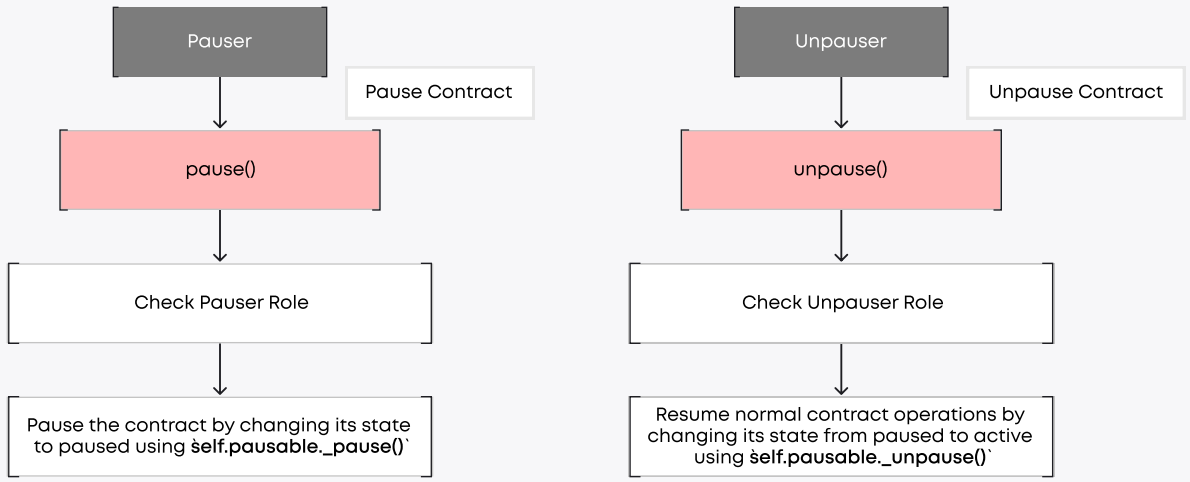


STARKDEFI (LOCKER)

Settings with Admin



Pause/unpause with Pauser/Unpauser



COMPLETE ANALYSIS

CRITICAL-1	✔ Resolved

NFT Operations Blocked.

NFT Operations Blocked.
StarkDLocker.cairo: withdraw(), extend_lock_duration(), transfer_lock(), split_lock().
The only_lock_receiver modifier assumes the validation of only ERC-20 locks from the locked_tokens, excluding the possibility of operating with NFT locks, which are stored in a separate mapping locked_nfts. Consequently, functions protected by this modifier are unable to process NFT locks, thereby blocking any operations involving NFT locks.

Recommendation:

Revise the only_lock_receiver modifier to properly handle both ERC-20 and ERC-721 locks by utilizing data from locked_tokens and locked_nfts.

Post-audit.

The approach to checking the receiver lock has been changed to properly handle both ERC-20 and ERC-721 locks.

CRITICAL-2	✔ Resolved

Missing Configuration Update.

StarkDLocker.cairo: enable_lock_fee, enable_nft_lock, configure_lock_fee.
In administrative functions the step to write the updated configuration back to the contract's state storage is missing. As a result, any changes to the configuration are not retained, meaning they do not take effect.

Recommendation:

Include a command to write the updated configuration data to the state storage after each change within the administrative functions.

Post-audit:
Configuration data is updated now.

MEDIUM-1	✔ Verified

No Limitations on the Number of Locks Per User.

StarkDLocker.cairo: _remove_from_user_lock_ids.
The _remove_from_user_lock_ids function iterates through an array in storage, which can become a performance issue if numerous locks are registered for a single user. The issue is marked as medium, since an attacker can transfer locks to other users, blocking operations where _remove_from_user_lock_ids is used.

Recommendation:

Introduce a limit on the maximum number of locks per user and consider optimizing the logic for managing lock arrays.

Post-audit:

The client team has verified that setting a limit on the number of locks per user is not the desired choice for the system's design. However, the audit team still expresses concern over this decision. Although removing such a limit offers greater flexibility, it introduces the risk of a user's account becoming overloaded with transferred locks. In scenarios where users attempt to transfer one of their many locks, they may face transaction failures due to large amounts of iterations required in _remove_from_user_lock_ids, which could lead to high gas costs and potentially reverted transactions. The audit team advises revisiting this design with these considerations.

MEDIUM-2	✔ Resolved

Logical Error in Amount Validation.

StarkDLocker.cairo: withdraw().
In the withdraw function the validation assertion `assert(amount.is_non_zero() || amount <= lock.amount, Errors::INVALID_AMOUNT);`contains a logical error where the usage of the 'or' (||) operator should instead be the 'and' (&&) operator. This could potentially allow attempts to withdraw a zero amount, leading to unnecessary gas consumption without actual token withdrawal.

Note: This issue is marked as medium severity and not critical because subsequent calculations and checks within the function will revert the transaction when an amount greater than the locked balance is provided. However, if the value 0`is passed, it will consume gas without completing any meaningful action.

Recommendation:

Update the incorrect logical OR operator to an AND operator in validation check.

Post-audit:

AND operator is used now in validation check.

LOW-1	✔ Resolved

No Validation for Lock Duration.

StarkDLocker.cairo: _check_lock_params(), extend_lock_duration(). The functions described above lack a validation for the unlock_time field against excessively large values, which could lead to tokens being locked for an unacceptably long duration. It would be sensible for the contract to have an upper bound for the lock duration, such as one year. There is also no validation for very short lock times.

Recommendation:

Verify the required range of unlock_time values and implement logic to cap the maximum and minimum lock duration to prevent indefinite or zero locking time.

Post-audit:
The maximum range unlock time has been verified and the minimum range logic was added.

LOWEST-1	✔ Resolved

Typos.
StarkDLocker.cairo: lock_token(), line 283-284: let trnasfer -> let transfer.

Recommendation:

Fix typos.

Post-audit:
Typo was fixed.

LOWEST-2	✔ Resolved

Lack of Event Emission.

StarkDLocker.cairo: configure_lock_fee(), whitelist_wallet(), enable_nft_lock(), enable_lock_fee().
In order to keep track of historical changes of storage variables, it is recommended to emit events on every change in the functions that modify the storage.

Recommendation:

Consider emitting events in all functions where state changes to reflect important changes in contract.

Post-audit:

Events emitted now.

LOWEST-3	✔ Resolved

Lack of Validation.

StarkDLocker.cairo: constructor() -> params fee_address, fee_token.
The zero address check is a standard validation to prevent initializing contracts without valid addresses. Add necessary checks to ensure that none of the addresses are equal to the zero address before setting them.

Recommendation:

Consider adding the necessary validation.

Post-audit:

Validation was added.

LOWEST-3.1	Unresolved

Lack of Validation.

StarkDLocker.cairo: configure_lock_fee() -> params fee, fee_token, fee_receiver.
All parameters described above do not have validation. Addresses are not checked for zero addresses, and fee amount are not checked for being zero amount.

Recommendation:

Consider adding the necessary validation.

Post-audit:

Validation was added, but the validation assertion for `fee_receiver` incorrectly uses `Errors::INVALID_TOKEN`, where it should reference `Errors::INVALID_RECEIVER` instead.

LOWEST-4	✔ Verified

User-Controlled Lock Period Adjustment.

StarkDLocker.cairo: lock_token(), extend_lock_duration().
The contract allows users to determine the duration of their token locks and to extend this duration. While this provides a great deal of autonomy to the user, it is important for the platform to ensure that this feature aligns with the overall operational and security model.
The issue is marked as Info, as it refers to the business logic (and authorized actions) solution choice and needs feedback from the team.

Recommendation:

Verify that the given functionality works as intended and lock period adjustment does not need the authorized control.

Post-audit:

Team verified that the given functionality works as intended.

LOWEST-5	✔ Resolved

Purpose for Whitelist Feature is Uncertain.

StarkDLocker.cairo: whitelist_wallet() and related storage whitelisted_wallets.
The contract contains a feature for maintaining a whitelist of wallets. The current implementation does not use this functionality and also does not clearly explain why it exists in the contract.

Recommendation:

Clarify the intended functionality of the whitelist feature. If the feature plays a role in planned future developments or has specific uses that are not immediately apparent, document this in contract. If the whitelist functionality is unnecessary, remove it.

Post-audit:
Functionality of the whitelist feature was clarified and added.

LOWEST-6	✔ Resolved

Unnecessary Initialization.

StarkDLocker.cairo: constructor() -> deposit_id.
In the constructor, the deposit_id variable is initialized to 0.
Because all uninitialized variables in Cairo are already set to 0 by default, this explicit initialization is redundant and leads to unnecessary gas consumption.

Recommendation:

Verify whether the initialization of deposit_id in the constructor is indeed necessary or consider removing it to optimize gas costs.
Fourth paragraph in `Your first contract` section:
https://docs.cairo-lang.org/hello_starknet/intro.html

Post-audit:
Initialization of deposit-id was removed.

LOWEST-7	✔ Resolved

Redundant Return Value Checks.

In OpenZeppelin contract implementations used in StarkDLocker, the return values from transfer functions are by default true, making assert checks for these calls unnecessary.

Recommendation:

Consider removing redundant return value checks from transfer functions to simplify the code and reduce gas costs.

Post-audit:

Redundant return value checks were removed.

LOWEST-8	✔ Verified

Unified deposit_id for ERC-20 and ERC-721 Tokens.

The StarkDLocker contract uses a single deposit_id counter for both ERC-20 and ERC-721 tokens. This might complicate frontend and backend integration, if more precise differentiation of locks is needed than the current implementation allows.

The issue is marked as Info, as it refers to the business logic solution choice and needs feedback from the team.

Recommendation:

Verify the need to separate identifiers for different token types and, if necessary, modify the contract for better management and tracking of locks.

Post-audit:

Team verified that now functionality works as intended and no need to change anything.

LOWEST-9	✔ Verified

Lock ID Not Removed from Arrays.

In OpenZeppelin contract implementations used in StarkDLocker, the return values from transfer functions are by default true, making assert checks for these calls unnecessary.

Recommendation:

Consider removing redundant return value checks from transfer functions to simplify the code and reduce gas costs.

Post-audit:

Redundant return value checks were removed.

LOWEST-10	✔ Resolved

Risk of Tokens Getting Stuck When Sent to Contract Unintentionally.

Based on the nature and the purpose of the contract the team should take into consideration the case that Users might accidentally (or purposely after misreading the flow and dApp interface) send ERC-20 or ERC-721 tokens directly to the contract's balance, resulting in a loss of access to these assets.

Issue is marked as Info, as it is related to the expected user behavior rather than contract functioning. So, the possible issue should be listed in the report and requires the feedback from the team, though the “rescue” mechanism is not mandatory.

Recommendation:

Verify that the dApp will have clear instructions on how users should interact with the protocol. Consider implementing a “rescue” mechanism that permits the contract owner or authorized individuals to retrieve tokens accidentally sent to the contract - though with the appropriate security checks regarding the balance stored via the legitimate flow.

Post-audit:

A function rescue_tokens() and rescue_nft() has been added to withdraw tokens.

LOWEST-11	✔ Resolved

Risk of Royalties Getting Stuck on Contract's Balance.

Some NFT tokens may generate royalties that could end up on the contract's balance when being returned to the user. Such royalty funds will be inaccessible or non-retrievable without an additional mechanism.

Note: for now the issue is marked as Hypothesis, which will be checked during the testing stage. Though the team may already leave comments.

Recommendation:

Add a mechanism to withdraw royalty funds stuck on the contract's balance.

Post-audit:

Auditors confirmed that stuck royalty funds could be withdrawn by rescue_tokens() if the royalty token doesn't match any locked assets - so locked assets are secured.

LOWEST-12	✔ Resolved

Inconsistency in Comment.

StarkDLocker.cairo: split_lock().

The function has a comment that specifies a return value (@return id of the new lock). However, the actual implementation of the function does not include a return statement, and therefore does not return any value. This discrepancy between comments and code implementation may lead to confusion for users and developers interacting with the contract interface.

Recommendation:

Review the `split_lock` function to clarify whether it should return a value. If a return value is intended, adjust the function implementation to include it. If not, update the comment to reflect the actual behavior of the function.

Post-audit:

Comment was removed.

LOWEST-13	✔ Resolved

Misleading amount display in Lock Details After Full Withdrawal.

StarkDLocker.cairo: withdraw().

When the entire balance of a lock is withdrawn the amount within the lock structure does not reflect this final state, as it is not set to 0. Although the lock's withdrawn status is marked as true, indicating a complete withdrawal, the non-zero amount displayed could mislead someone analyzing the lock's remaining balance.

Recommendation:

Consider revising the `withdraw` function to set `amount` to `0` in the lock's data structure when the full amount has been withdrawn to provide a clear and accurate representation of the lock's final state in the contract.

Post-audit:

Now the amount is set to zero in the lock's data structure when the full amount has been withdrawn.

STANDARD CHECKLIST

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

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

test starkdefi::tests::locker::test_locker::test_constructor_locker ... ok (gas usage est.: 3292620)
test starkdefi::tests::locker::test_locker::test_deployed_locker ... ok (gas usage est.: 2278140)
test starkdefi::tests::locker::test_locker::test_rescue_tokens_invalid_token ... ok (gas usage est.: 4861150)
test starkdefi::tests::locker::test_locker::test_unpause ... ok (gas usage est.: 8914440)
test starkdefi::tests::locker::test_locker::test_upgrade ... ok (gas usage est.: 4311620)
test starkdefi::tests::locker::test_locker::test_upgrade_invalid_hash ... ok (gas usage est.: 2033760)
test starkdefi::tests::locker::test_locker::test_transfer_lock ... ok (gas usage est.: 12101200)
test starkdefi::tests::locker::test_locker::test_lock_token ... ok (gas usage est.: 11444050)
test starkdefi::tests::locker::test_locker::test_rescue_tokens ... ok (gas usage est.: 6591480)
test starkdefi::tests::locker::test_locker::test_lock_token_invalid_token ... ok (gas usage est.: 4496050)
test starkdefi::tests::locker::test_locker::test_transfer_lock_invalid_receiver ... ok (gas usage est.: 9324680)
test starkdefi::tests::locker::test_locker::test_lock_token_invalid_amount ... ok (gas usage est.: 4496050)
test starkdefi::tests::locker::test_locker::test_lock_token_invalid_receiver ... ok (gas usage est.: 4496050)
test starkdefi::tests::locker::test_locker::test_rescue_nft_with_royalties ... ok (gas usage est.: 10244800)
test starkdefi::tests::locker::test_locker::test_transfer_lock_withdrawn_already ... ok (gas usage est.: 10796210)
test starkdefi::tests::locker::test_locker::test_lock_token_invalid_unlock_time ... ok (gas usage est.: 4502850)
starkdefi::tests::locker::test_locker::test_scenario_lock_withdraw_transfer_lock_whitelist_acc2 ... ok (gas usage est.: 18779280)
test starkdefi::tests::locker::test_locker::test_enable_lock_fee ... ok (gas usage est.:

test starkdefi::tests::locker::test_locker::test_enable_nft_lock ... ok (gas usage est.: 3112010)
test starkdefi::tests::locker::test_locker::test_whitelist_wallet ... ok (gas usage est.: 2802230)
test starkdefi::tests::locker::test_locker::test_rescue_locked_nft ... ok (gas usage est.: 7168450)
test starkdefi::tests::locker::test_locker::test_configure_lock_fee ... ok (gas usage est.: 3120710)
test starkdefi::tests::locker::test_locker::test_configure_lock_fee_zero_amount ... ok (gas usage est.: 2315250)
test starkdefi::tests::locker::test_locker::test_lock_nft ... ok (gas usage est.: 8785420)
test starkdefi::tests::locker::test_locker::test_configure_lock_fee_zero_token ... ok (gas usage est.: 2315250)
test starkdefi::tests::locker::test_locker::test_rescue_tokens_token_already_locked ... ok (gas usage est.: 10771890)
test starkdefi::tests::locker::test_locker::test_pause ... ok (gas usage est.: 7048920)
test starkdefi::tests::locker::test_locker::test_scenario_transfer_lock_loop ... fail (gas usage est.: 27015240)
test starkdefi::tests::locker::test_locker::test_withdraw ... ok (gas usage est.: 13817160)
test starkdefi::tests::locker::test_locker::test_withdraw_invalid_zero_amount ... ok (gas usage est.: 9002230)
test starkdefi::tests::locker::test_locker::test_scenario_nft_withdraw_transfer_lock_extend_duration ... ok (gas usage est.: 14000160)
test starkdefi::tests::locker::test_locker::test_extend_lock_duration_invalid_time ... ok (gas usage est.: 8839200)
test starkdefi::tests::locker::test_locker::test_withdraw_nft ... ok (gas usage est.: 10724530)
test starkdefi::tests::locker::test_locker::test_split_lock_invalid_unlock_time ... ok (gas usage est.: 9348450)
test starkdefi::tests::locker::test_locker::test_withdraw_withdrawn_already ... ok (gas usage est.: 10463190)

```
test starkdefi::tests::locker::test_locker::test_split_lock ... ok (gas usage est.: 12706180)
test starkdefi::tests::locker::test_locker::test_withdraw_invalid_amount ... ok (gas usage est.: 9002230)
test starkdefi::tests::locker::test_locker::test_split_lock_invalid_amount ... ok (gas usage est.: 9351220)
test starkdefi::tests::locker::test_locker::test_withdraw_still_locked ... ok (gas usage est.: 9978270)
test starkdefi::tests::locker::test_locker::test_split_lock_nft ... ok (gas usage est.: 7952750)
test starkdefi::tests::locker::test_locker::test_split_lock_withdrawn_already ... ok (gas usage est.: 10817650)
test starkdefi::tests::locker::test_locker::test_extend_lock_duration ... ok (gas usage est.: 11721660)
test starkdefi::tests::locker::test_locker::test_extend_lock_duration_withdrawn_already ... ok (gas usage est.: 10308400)

failures:
   starkdefi::tests::locker::test_locker::test_1scenario_transfer_lock_loop - Panicked with (0x4f7574206f6620676173 ('Out of gas'), 0x454e545259504f494e545f4641494c4544 ('ENTRYPOINT_FAILED'), 0x454e545259504f494e545f4641494c4544 ('ENTRYPOINT_FAILED')).

Error: test result: FAILED. 42 passed; 1 failed; 0 ignored.
```

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.