Blaize.Security

LFIT
SMART CONTRACT AUDIT

August 30th, 2024 / V. 2.0

Table of Contents

Executive Summary

During the audit, we examined the security of the LFIT smart contract for the Klaytn blockchain. Our task was to find and describe any security issues in the platform's smart contract. This report presents the security audit findings conducted between June 19, 2024, and June 21, 2024.

During the audit, the Blaize Security team audited the LFIT contract. The LFIT is a regular ERC20 contract enhanced with an owner-side token-locking mechanism. More information can be found in the protocol overview section. Therefore, the auditors' goal was to analyze the security of funds and the correctness of the ERC20 standard implementation for the Klaytn blockchain - these and other potentially vulnerable areas (including several internal checklists).

During the audit, the auditors identified two low and a few information issues. Issues were connected to absence of restrictions on the owner's activities with users' tokens, parameter validations, best practices support. Currently, issues are not addressed.

Also, auditors should note several points:

1. The project utilizes the solc 0.5.12 (Istanbul), which is currently officially supported by the Klaytn EVM.
2. The contract has a high risk of centralization and single point-of-failure. The owner can pause/freeze transfer operations for all users (or any particular) at any moment without restrictions. The owner can lock any user's balance - again without any restrictions. Thus, if the owner's private key is compromised, the malicious actor will be able to ruin the contract's activity. So, the contract is open to both types of risks, which is a negative security factor. The team should provide a high level of security for the owner private key, with the usage of multisigs and/or cold-wallet approaches.

The auditors also noted negative factors for the project quality: a lack of natspec comments, a lack of general documentation, and an absence of tests. Nevertheless, the overall security of the smart contract is high enough—despite several unresolved issues. The project passed the audit with no critical or high-risk issues, though it still has room for improvement and several unresolved issues from auditors.

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 / roles structure review, analysis of user and admin behavior;
- Review of dependencies, 3rd parties, and integrations;
- Review with automated tools and static analysis;
- Vulnerabilities analysis against several checklists, including internal Blaize.Security one;
- Storage usage review;
- Gas (or tx weight or cross-contract calls or another analog) optimization;
- Code quality, documentation, and consistency review.

TESTING STAGE:

- Development of edge cases from 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.

Audit Rating

Score: 9.2 /10

	RATING
Security	9.2
Logic optimization	9.7
Code quality	9.7
Testing suite	0
Documentation	9

Security: General mark for the security of the protocol.
The main mark for the audit qualification.

Logic optimization: Evaluation of how optimal the implementation is, including presence of extra/unused code, uncovered/extra cases, gas (or its analog) optimization, memory management optimization, etc
Code quality: Evaluation of best practices followed, code readability, structure and convenience of further development
Testing suite: Availability of the native tests suite, level of logic coverage, checks of critical areas being covered.
Documentation: Availability and quality of the documentation, coverage of core functionality and user flows: whitepaper, gitbook, readme, specs, natspec, comments in the code and other possible forms of documentation.

SECURITY RATING CALCULATION

Approximate weight of unresolved issues.
Critical: -3 points
High: -2 points
Medium: -0.5 points
Low: -0.1 points
Informational: -0.1 point (in general, depends on the context)

Note: additional concerns, violated checklist items (including standard vulnerabilities), and verified backdoors may influence the final mark and weight of certain issues.

Starting with a perfect score of 10:

Critical issues: 0 issue (0 resolved): 0 points deducted

High issues: 0 issue (0 resolved): 0 points deducted

Medium issues: 0 issue (0 resolved): 0 points deducted

Low issues: 2 issues (0 resoled): -0.2 points deducted

Informational issues: 3 issues (0 resolved: -0.3 points deducted

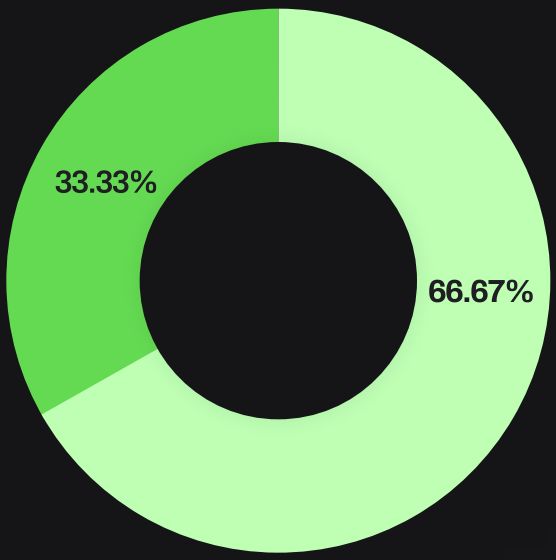
-0.1 points deducted for the absence of tests
-0.2 points deducted for the centralization and single point of failure risks

Security rating = 10 - 0.2 - 0.3 - 0.1 - 0.2 = 9.2

Technical Summary

THE GRAPH OF VULNERABILITIES DISTRIBUTION:

- Critical
- High
- Medium
- Low
- Info



The table below shows the number of the detected issues and their severity. A total of 5 problems were found. 5 issues were fixed or verified by the Customer's team.

	FOUND	FIXED/VERIFIED
Critical	0	0
High	0	0
Medium	0	0
Low	2	0
Info	3	0

SEVERITY DEFINITION

CRITICAL

The system contains several issues ranked as very seriousand 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.

INFO

The issue has no impact on the contract's ability to operate, yet is relevant for best practices. Or this status can be assigned to the issues related to the suspicious activity or substandard business logic decisions which cannot be classified without the comments from the team (and can be re-classified on the later audit stages).

Issues reviewed by the team can get the next statuses:
Resolved: issue is resolved by an appropriate patch or changes in the business logic
Verified: the team provided sufficient evidences that the issue describes desired behavior
Unresolved: neither path nor comments provided by the team, or they are not sufficient to resolve the issue
Acknowledged: the team accepts the misbehavior and connected risks

Audit Scope

Language/Technology: **Solidity**

Blockchain: **Polygon**

The scope of the project includes:

- contracts\LFIT.sol

The source code of the smart contract was taken from the branch: **master**.
Initial commit:

- d58a614996f784802340476e2b244a21c5b90144

Final commit:

- 71457e298964ad9045c7359740c01a4f162a1caf

After the audit repository was renamed to:
https://github.com/lfit-official/lfit_erc20
And the target chain was changed from Klaytn to Polygon

Protocol overview

DESCRIPTION

The contract represents the token of the KIP7 standard (Klaytn analog of ERC20 standard) with additional functionality:

- The token is burnable; thus, any user can burn tokens from their own balance and can burn allowed tokens from other balances. The burn interface is public;
- The token is pausable - thus, the owner can pause the transfer and approve functionality;
- The token contains freeze functionality - the owner of the contract freeze/unfreeze the account of any user, thus pausing transfer functionality for certain users;
- Token lock functionality. The owner can lock any user's tokens from their balances for a certain period. After the lock period is finished, tokens are automatically unlocked during the next transfer.

Inheritance tree:

0) LFIT.sol <- Ownable, KIP7Burnable, KIP7Pausable, KIP7Metadata
- contains lock logic
- contains overridden transfer methods to include locks during transfer
- contains freeze/unfreeze logic
- contains mint during the construction

1.1) KIP7Metadata <- KIP13
- contains name, symbol and decimal fields for the token

1.2) KIP7Burnable <- KIP13, KIP7
- contains public interface for burn() and burnFrom()

1.3) KIP7Pausable <- KIP13, KIP7, Pausable
- overloaded transfer/approve functions with restrictions for the paused protocol.

2.1) Pausable <- Ownable
- contains “paused” flag available for the owner and functions to trigger it

2.2) KIP7 <- KIP13:
- ERC20 interface

3.1) Ownable: saves an owner address and ownership transfer/renounce functions

3.2) KIP13: analog of the ERC165

ROLES AND RESPONSIBILITIES

1. Owner (from the Ownable contract):
- is set during the deployment
- can pause/unpause all transfer/approve operations.
- can freeze/unfreeze any account.
- can lock any amount of token for any user, for any period at any moment of time

2. Regular user
- interacts with regular ERC20 functionality

LIST OF VALUABLE ASSETS

LFIT token itself
- can be used as a regular ERC20 token
- can be locked on the user's balance for certain period of time
- the whole supply of 3 bln tokens is minted during the construction

Contract is not supposed to hold any asset, thus any assets transferred directly to the contract will be lost.

SETTINGS

- owner of the contract (set during construction)
- ERC20 details (name, ticker, decimals) - set to ('LFIT', 'LFIT', 18) during construction
- pause flag - set to false during construction

DEPLOYMENT SCRIPT

No deployment scripts provided. However, the contract is delivered in a form of a flattened contract with all values defaulted in the constructor. Therefore the deployment script should be trivial in this case and can be avoided in a favour of the manual deployment.

Complete Analysis

STANDARD CHECKLIST / VULNERABLE AREAS

Storage structure and data modification flow	Pass
Access control structure, roles existing in the system	Pass
Public interface and restrictions based on the roles system	Pass
General Denial Of Service (DOS)	Pass
Entropy Illusion (Lack of Randomness)	N/A
Order-dependency and time-dependency of operations	Pass
Accuracy loss, incorrect math/formulas other violated operations with numbers	Pass
Validation of function parameters, inputs validation	Pass
Asset management, funds flow and assets conversions	Pass
Signatures replay and multisig schemes security	Pass
Asset Security (backdoors connected to underlying assets)	Pass
Incorrect minting, initial supply or other conditions for assets issuance	Pass
Global settings mis-using, incorrect default values	Pass
Violated communication between components/modules, broken co-dependencies	N/A
3rd party dependencies, used libraries and packages structure	N/A
Single point of failure	Fail
Centralization risk	Fail
General code structure checks and correspondence to best practices	Pass
Language-specific checks	Pass

LOW-1 Unresolved

MISSING BOUNDARIES VALIDATION FOR THE PARAMETER

lock(): there are no restrictions on the owner's actions. Thus, the owner can lock any amount for any period at any moment in time. Therefore, it is recommended:

- to have min/max boundaries for the lock period (difference between now and the releaseTime) so that the lock is not set for the unreachable amount of time
- to have the min/max boundaries for the amount to be locked, to avoid locking the full amount, or creating a DDoS opportunity with 0 amount locks

It is recommended to have several validations for each parameter:

- secure default value
- min and max boundaries
- extreme values (in case boundaries are not introduced)

Though no default value is set (including deployment/tuning/upgrade scripts), and there are no min/max values to validate against, the parameter is exposed to the risk of misconfiguration and/or human error.
The issue is marked as Low, as the authorized role controls the parameters despite the risk of misconfiguration. However, changing the incorrect parameter may take some time, thus creating a window of instability.

RECOMMENDATION:

Provide the default value for the parameter, introduce reasonable min/max boundaries for the parameter.

POST-AUDIT:

Currently issue is not addressed.

LOW-2 Unresolved

DDOS OPPORTUNITY FOR THE AMOUNT OF LOCKS

Currently, the function has no restrictions on the number of locks to be created for the user. Meanwhile, the _autoUnlock() function traverses all users' locks. So, after a certain number of locks are created for the user, the function will fail at an out-of-gas error.

The issue is marked as Low, as the locks creation functionality is available for the owner only, and it takes around 700 locks to make it unavailable for a certain user

RECOMMENDATION:

Provide a restriction for the number of locks that can be created for the user. Or create an additional public unlock function that traverses a limited number of lock records (or unlocks only a certain one).

POST-AUDIT:

Currently issue is not addressed.

INFO-1 Verified

POSSIBLE TRANSFER FOR THE FROZEN ACCOUNT

The contract restricts the sender not to be frozen, though it is still possible to transfer tokens TO the frozen account. Such behavior is substandard, thus requires verification from the team.

The issue is marked as Info, as it is connected to the business logic decision and cannot be classified without comments from the team.

RECOMMENDATION:

Verify, that the frozen account still can receive tokens or restrict the receiver.

POST-AUDIT:

Currently issue is not addressed.

INFO-2 Resolved

BURN FUNCTIONALITY IS NOT RESTRICTED

Currently the burn functionality is in the public interface of the contract and is not restricted. While it is a regular practice, still it requires verification from the team. Non-restricted burn has some risks for the healthy circulation supply and creates a risk of accidental burn of user's tokens.

The issue is marked as Info, as it is connected to the business logic decision and requires verification from the team.

RECOMMENDATION:

Verify that the burn functionality should not be restricted and aligns with the desired business logic

POST-AUDIT:

Currently issue is not addressed.

INFO Resolved

BEST PRACTICES AND CODE STYLE VIOLATIONS, OPTIMIZATIONS

1) LFIT.sol,lock().
Add a restriction against the 0 amount to avoid the creation of empty locks - or follow the recommendations from Low-1 issue.

2) Unused return values: unfreezeAccount(), freezeAccount(), lock(), transferWithLock(), _lock(), _unlock(), _autoUnlock()

- functions always return “true”, making the return value uniformational. Since there is no need to have a return value for these functions, it is recommended to remove it and simplify the interface.

3) _unlock(): update the array only if its length is greater than 1 to save some gas.

```
delete timelockList[holder][idx];
if (timelockList[holder].length > 1) {
    timelockList[holder][idx] = timelockList[holder][timelockList[holder].length.sub(1)];
}
timelockList[holder].length -=1;
```

RECOMMENDATION:

Consider correction of all listed issues for increasing the code readability and logic optimization

POST-AUDIT:

Currently issue is not addressed.

CODE COVERAGE AND TEST RESULTS FOR ALL FILES, PREPARED BY BLAIZE SECURITY TEAM

Successful deployment
Successful transfer
Successful transfer with locked asset
Successful transfer with auto unlock
Should revert when the sender is frozen
Successful transfer from
Successful transfer with locked asset
Should revert when the sender is frozen
Successful freeze account
Should revert when non-owner tries to call the function
Should revert when the account is already frozen
Successful unfreeze account
Should revert when non-owner tries to call the function
Should revert when the account is non-frozen
Successful lock tokens
Should revert when non-owner tries to call the function
Should revert when there is not enough balance
Should revert when release time is the past
Successful transfer
Should revert when non-owner tries to call the function

RESULTING COVERAGE

FILE	% STMTS	% BRANCH	% FUNCS
LFIT.sol	100	97.5	100

NATIVE TESTS OVERVIEW

The LFIT team did not provide native tests.

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.