HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Lith LCC
Date: Jun 29th, 2022

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for Lith LCC
Approved By	Evgeniy Bezuglyi | SC Department Head at Hacken OU Noah Jelich | Senior Solidity SC Auditor at Hacken OU
Type	ERC20 Upgradeable Token, Migration, Fee Distribution
Platform	EVM
Network	Ethereum
Language	Solidity
Methods	Manual Review, Automated Review, Architecture review
Website	https://www.lithtoken.io/
Timeline	17.05.2022 - 29.06.2022
Changelog	25.05.2022 - Initial Review 07.06.2022 - Second Review 29.06.2022 - Third Review

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by Lith LCC (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

Scope

The scope of the project is smart contracts in the repository:

Initial review scope

Repository:
https://github.com/LITHToken/LITx-Open/
Commit:
8c8c60d
Technical Documentation:
Type: Whitepaper (partial functional requirements provided)
Link: https://www.lithtoken.io/hubfs/LITH-Token-Whitepaper.pdf

Type: Functional requirements
Link: No, provided in whitepaper

Integration and Unit Tests: Yes
Contracts:
File: ./contracts/FeeDistributor.sol
SHA3: f2f1f43bafeeb1b35c8015374100358893c0b3c02a37c8023eea3a88560d5678

File: ./contracts/LITx.sol
SHA3: 599134d37cd276e37ed9fcffde64cd63055054532b7f8bcc2e45632ca09aae39

File: ./contracts/utils/BannedUpgradeable.sol
SHA3: a749028c51808e1a3fcf8bd8aa668e4bc0cf0318ddb09767c018e2e09c39eb62

File: ./contracts/utils/LITh.sol
SHA3: 07531f0a7d4aaa9e3b1f53248e78c15dcd05ae9136c371ac51ac2a78e84c34d3

File: ./contracts/utils/MerkleDistributor.sol
SHA3: 27519255f0d24c2ee76f488a2615e64a00068881d5ec578faaa1e9d4e9682b45

Second review scope

Repository:
https://github.com/LITHToken/LITx-Open/
Commit:
e8e9a89
Technical Documentation:
Type: Whitepaper (partial functional requirements provided)
Link: https://www.lithtoken.io/hubfs/LITH-Token-Whitepaper.pdf

Type: Functional requirements
Link: No, provided in whitepaper

Integration and Unit Tests: Yes
Contracts:
File: ./contracts/FeeDistributor.sol
SHA3: f848b39cbc97c61a2aa0ed18c340a7c1d3e495f66d7fcdaee73504130f60dbf6

File: ./contracts/LITx.sol

SHA3: 43a507ec15841e120ce40af4aa11ab1d127d4ed64d4b6fcbe6e7a2ac55f1b1dc

File: ./contracts/utils/BannedUpgradeable.sol
SHA3: 1a6eb70459332ca1841a7a0e6317ffde6f7a582d7e72479eb14623031efde4b5

File: ./contracts/utils/LITh.sol
SHA3: dad505270f0c77c580444ae76540c7f70f4031a7bc98b70f7994fc9575691cd8

File: ./contracts/utils/MerkleDistributor.sol
SHA3: 5425b4334795afb426e679a2651d4494eaec4f02b3a7be00b2b04914e5582780

Third review scope

Repository:
https://github.com/LITHToken/LITx-Open/
Commit:
e03e497
Technical Documentation:
Type: Whitepaper (partial functional requirements provided)
Link: https://www.lithtoken.io/hubfs/LITH-Token-Whitepaper.pdf

Type: Functional requirements
Link: No, provided in whitepaper

Integration and Unit Tests: Yes
Contracts:
File: ./contracts/FeeDistributor.sol
SHA3: c3bcf2db42efa775dc1da0068e66ee91eb267eed2a45b1c1c9f4bf7c19dd945d

File: ./contracts/LITx.sol
SHA3: 4d02295f9f9a1336311984a2919223e7721a8a27447e6cc086d61519ad952fc2

File: ./contracts/utils/BannedUpgradeable.sol
SHA3: 1a6eb70459332ca1841a7a0e6317ffde6f7a582d7e72479eb14623031efde4b5

File: ./contracts/utils/LITh.sol
SHA3: dad505270f0c77c580444ae76540c7f70f4031a7bc98b70f7994fc9575691cd8

File: ./contracts/utils/MerkleDistributor.sol
SHA3: 5425b4334795afb426e679a2651d4494eaec4f02b3a7be00b2b04914e5582780

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions.
Medium	Medium-level vulnerabilities are important to fix; however, they cannot lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Executive Summary

The score measurement details can be found in the corresponding section of the methodology.

Documentation quality

The Customer provided both functional and technical requirements. The total Documentation Quality score is **10** out of **10**.

Code quality

The total CodeQuality score is **8** out of **10**. Follows official language style guides. Incomplete test coverage - missing negative tests.

Architecture quality

The architecture quality score is **10** out of **10**. Clean and clear architecture.

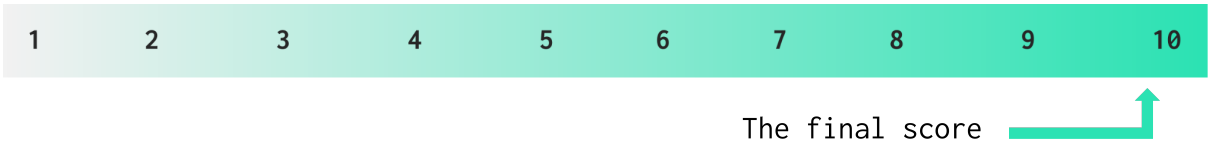
Security score

As a result of the audit, the code contains **1** low severity issue. The security score is **10** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **9.8**.



Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Checked Items

We have audited provided smart contracts for commonly known and more specific vulnerabilities. Here are some of the items that are considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Not Relevant
Access Control & Authorization	CWE-284	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	SWC-106	The contract should not be destroyed until it has funds belonging to users.	Not Relevant
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Uninitialized Storage Pointer	SWC-109	Storage type should be set explicitly if the compiler version is < 0.5.0.	Not Relevant
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Not Relevant
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Passed
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless it is required.	Passed
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed

Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Not Relevant
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Not Relevant
Signature Unique Id	SWC-117 SWC-121 SWC-122	Signed messages should always have a unique id. A transaction hash should not be used as a unique id.	Not Relevant
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes.	Not Relevant
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Not Relevant
Presence of unused variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Passed
EIP standards violation	EIP	EIP standards should not be violated.	Passed
Assets integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions.	Passed
User Balances manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	**Custom**	Smart contract data should be consistent all over the data flow.	Passed
Flashloan Attack	**Custom**	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant
Token Supply manipulation	**Custom**	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the customer.	Passed
Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed

Style Guide Violation	**Custom**	Style guides and best practices should be followed.	Passed
Requirements Compliance	**Custom**	The code should be compliant with the requirements provided by the Customer.	Passed
Repository Consistency	**Custom**	The repository should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Tests Coverage	**Custom**	The code should be covered with unit tests. Test coverage should be 100%, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Failed
Stable Imports	**Custom**	The code should not reference draft contracts, that may be changed in the future.	Passed

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

System Overview

LITh is a mixed-purpose system with the following contracts:

- *LITx* - upgradeable ERC-20 token that mints all initial supply to the **token contract**. Additional minting is **allowed**. Takes a **1% fee** from every transfer.

  It has a **ban system,** and banned users cannot migrate their $LITh tokens, and a **migration mechanism** between $LITh to $LITx token with a 1:1 rate.

  Will transfer **all remaining balance** on the contract to the ecosystem named address after the specified time, which will be set during deployment, pass, and will **no longer support migrations.**

  It has **bridge support** and can **mint** the desired amount of tokens according to the **bridge order**. Security of the bridge is **not** included in the audit scope.

  It has the following attributes:
  - Name: LITx token
  - Symbol: LITx
  - Decimals: 18
  - Total supply: 5.417.770.823

- *LITh* - upgradeable ERC20 token that mints all initial supply to the deployer. It is used for simulating $LITh to $LITx migration.
  It has the following rates to distribute tokens:
  - Name: LITh token
  - Symbol: LITh
  - Decimals: 18
  - Initial Supply: 100.000

- *MerkleDistributor* - an abstract contract for distributing tokens to predefined addresses using the Merkle Tree Algorithm.

- *FeeDistributor* - upgradeable contract used for distributing **$LITx** fees collected from user transfers. It is planned to distribute the collected fees in 4 separate titles: developer, ecosystem, marketing, and reward.

  Uses *MerkleDistributor contract* to verify whether the reward is collected or not. If the reward has not been collected within thirty days, the uncollected amount carries over to the next round of the reward distribution. It is **not** within the scope of this audit whether the **rewards** will be distributed fairly and whether the **ecosystem** and **marketing** shares will be spent for their own purposes.

It uses the following ratios for token distribution:
  - Ecosystem: 60%
  - Marketing: 20%
  - Developers: 10%
  - Rewards: 10%

- *BannedUpgradeable* - an abstract contract that implements a ban system. It has the properties of banning and unbanning addresses.

Privileged roles

- The owner of the LITx contract has the following privileges:
  - Can change fee distributor contract address.
  - Can ban and unban an address.

- The bridge privilege can mint the desired amount of $LITx to any address.

- The owner of FeeDistributor contract has the following privileges:
  - Can change the developer addresses.
  - Can change Merkle root, and if not claimed, rewards are left with the previous root to be used for next month's rewards. Users cannot claim their rewards anymore.
  - Can distribute collected fees to developers, ecosystem, and marketing addresses with the mentioned ratio.

Risks

- The amount of the reward to be distributed is not controlled on chains. It is done according to the Merkle proof set by the owner. A misrepresentation of the proof may result in an under-or over-distribution of rewards.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Findings

Critical

No critical severity issues were found.

High

1. **Owners can distribute user rewards**

   In the FeeDistributer contract, when “distribute” is called, the user allocated rewards are part of the pool being distributed.

   This may cause users to be unable to collect their rewards due to an insufficient contract balance.

   File: ./contracts/FeeDistributor.sol

   Contract: FeeDistributor

   Function: distribute

   Recommendation: Change the distribution logic so it does not depend on the contract balance at the time of distribution.

   Ensure there is always enough balance for the users to claim rewards by keeping the amount protected on the contract.

   Status: Fixed (Revised commit: e8e9a89)

2. **Banned users can still transfer tokens**

   The ban system is implemented on LITx contract by adding “nonBanned” modifier to the “_transfer” function, but the ban system only checks the sender of the message.

   This means an address can still transfer its funds using approval.

   File: ./contracts/LITx.sol

   Contract: LITx

   Functions: _transfer

   Recommendation: Edit the "nonBanned" modifier and give both sender and recipient addresses as input parameters on the transfer function.

   Status: Fixed (Revised commit: e8e9a89)

3. **Cannot claim rewards in case of root change**

   In the FeeDistributer contract, the owner can change Merkle Root. However, if users do not claim rewards before root change, it is impossible to claim after the change operation.

   This issue may cause users not to be able to collect their rewards.

   File: ./contracts/FeeDistributor.sol

   Contract: FeeDistributor

Recommendation: Migrate to a solution that only updates the max claimable amount, instead of having atomic claims that depend on a specific MerkleRoot - e.g. a signature-based claimable balance update.

Status: Fixed (Revised commit: e8e9a89)

Medium

1. No return value check for token transfers

ERC20 transfer functions return bool after transfers, and it is important to implement a return value check for this return value.

This issue leads to unintended behavior of contract about token transfer result.

Files: ./contracts/LITx.sol, ./contracts/FeeDistributor.sol

Contracts: LITx, FeeDistributor

Functions: migrate, claim, _distributeDevelopers

Recommendation: Implement a return value check for token transfers.

Status: Fixed (Revised commit: e03e497)

Low

1. Unlocked pragma

Unlocked pragmas may cause the contract to be deployed with a different Solidity version from the tested.

This can lead to encountering undiscovered bugs.

Files: ./contracts/LITx.sol, ./contracts/FeeDistributor.sol, ./contracts/utils/MerkleDistributor.sol, ./contracts/utils/LITh.sol, ./contracts/utils/BannedUpgradeable.sol

Contracts: LITx, FeeDistributor, MerkleDistributor, LITh, BannedUpgradeable

Recommendation: Lock pragma to a specific compiler version.

Status: Fixed (Revised commit: e8e9a89)

2. Redundant variables

“reward” and “REWARD” variables have no implementation on the FeeDistributor contract.

Keeping redundant variables increases Gas costs during deployment.

Files: ./contracts/FeeDistributor.sol

Contract: FeeDistributor

Recommendation: Remove redundant variables.

Status: Fixed (Revised commit: e8e9a89)

3. **Missing zero address validation.**

   Address parameters are being used without checking against the possibility of 0x0.

   This can lead to unwanted external calls to 0x0.

   Files: ./contracts/FeeDistributor.sol, ./contracts/LITx.sol

   Contracts: FeeDistributor, LITx

   Functions:__FeeDistributor_init_unchained, __LITxToken_init_unchained

   Recommendation: Implement zero address checks

   Status: Fixed (Revised commit: e8e9a89)

4. **Usage of public visibility instead of external**

   “setMerkleRoot”, “distribute”, “setDevelopers” and “initialize” functions in FeeDistributor contract and “initialize” function in LITx contract and “initialize” function in LITh contract are declared as public but are not called by contracts.

   Files: ./contracts/utils/LITh.sol, ./contracts/FeeDistributor.sol, ./contracts/LITx.sol

   Contracts: LITh, FeeDistributor, LITx

   Functions: setMerkleRoot, distribute, setDevelopers, initialize

   Recommendation: Convert to external. Making functions external instead of public reduces Gas costs during execution.

   Status: Fixed (Revised commit: e8e9a89)

5. **Incorrect usage of memory pointer**

   In LITx contract “initialize”, “__LITxToken_init”, “__LITxToken_init_unchained” functions’ pointer of “chains_” parameter is memory, but it should be calldata.

   File: ./contracts/LITx.sol

   Contract: LITx

   Functions: initialize, __LITxToken_init, __LITxToken_init_unchained

   Recommendation: Change memory pointers to calldata.

   Status: Fixed (Revised commit: e8e9a89)

6. **Not possible to support future chains**

   LITx contract is keeping supported chains with a "mapping" and those are being set during deployment. However, there is no function to add new chains to the "mapping".

   With the current version of the contract, it is impossible to support different chains other than the set ones during deployment.

File: ./contracts/LITx.sol

Contract: LITx

Recommendation: Add a function to the contract to add new chains to the "chains" mapping.

Status: Fixed (Revised commit: e8e9a89)

7. **Redundant nonBanned modifier usage**

   Usage of the *nonBanned* modifier is redundant. The *_transfer* function, called later, already has this modifier.

   File: ./contracts/LITx.sol

   Contract: LITx

   Function: migrate

   Recommendation: Remove redundant modifier usage.

   Status: Reported

8. **Redundant usage of SafeERC20**

   In the LITx contract “migrate” function, "safeTransferFrom" method is used to transfer $LITh token and in the FeeDistributer contract, ”claim”, “distribute” and “_distributeDevelopers” functions, “safeTransfer” method is used to transfer $LITx token.

   Since both $LITh and $LITx tokens conform to the ERC20 standard, it is not required to use “safeTransferFrom” and “safeTransfer” functions nor the SafeERC20 contract.

   Files: ./contracts/LITx.sol, ./contracts/FeeDistributor.sol

   Contracts: LITx, FeeDistributor

   Functions: migrate, claim, _distributeDevelopers

   Recommendation: Remove SafeERC20 import and its implementations, use ERC20 transfers and implement a return value check.

   Status: Fixed (Revised commit: e8e9a89)

9. **Redundant usage of reentrancy guard**

   In the LITx contract “migrate” function, the reentrancy guard is used. However, the “migrate” function makes one external function call to the "migrateToken" contract, which is the $LITx token and thus a trusted address. It is not required to use a reentrancy guard in this case.

   File: ./contracts/LITx.sol

   Contract: LITx

   Function: migrate

   Recommendation: Remove the reentrancy guard from the contract.

Status: Fixed (Revised commit: e8e9a89)

10. Redundant import

SafeERC20Upgradeable contract is imported in FeeDistributor and LITx contracts; however, not implemented in this contract because not needed.

Files: ./contracts/LITx.sol, ./contracts/FeeDistributor.sol

Contract: LITx, FeeDistributor

Recommendation: Remove *SafeERC20Upgradeable* import.

Status: Fixed (Revised commit: e03e497)

11. Redundant variable

MerkleRootChanged event is created in FeeDistributor contracts but not used. It is required to emit events on critical state changes; therefore, *MerkleRootChanged* event can be emitted in *distribute* function.

Files: ./contracts/FeeDistributor.sol

Contract: FeeDistributor

Recommendation: Emit *MerkleRootChanged* event in *distribute* function.

Status: Mitigated (with customer notice)

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed by the best industry practices at the date of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The audit makes no statements or warranties on the security of the code. It also cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the audit cannot guarantee the explicit security of the audited smart contracts.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

B: Lith LLC - Recommended Actions

Next Steps

1. Identify critical assets to scope. We recommend to scope 5 assets:
    - ERC-20 Smart Contract
    - https://lithtoken.io
    - iOS app
    - Android app
    - Network
2. Pentest 4 assets:
    - https://lithtoken.io
    - iOS app
    - Android app
    - Network
3. Set up crowdsourced security for all assets:
    - ERC-20 Smart Contract
    - https://lithtoken.io
    - iOS app
    - Android app
    - Network