HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: LAK3
Date: 28 Sep, 2023

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 LAK3
Approved By	Paul Fomichov | Lead Solidity SC Auditor at Hacken OU
Type	ERC20 token; Vesting; MultiSig
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://lak3.io/
Changelog	08.09.2023 - Initial Review 28.09.2023 - Second 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 LAK3 (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.

System Overview

The LAKE project encompasses an ERC20 token-based smart contract system, integrated within a Web3 ecosystem, with the objective of decentralizing access to water resources with the following contracts:

- *LakeToken* – simple ERC-20 token. Additional minting is allowed. It has the following attributes:
  - Name: LAKE
  - Symbol: LAK3
  - Decimals: 18
  - Total supply: 950m tokens.
- *LakeVestingAdvisors* – a token vesting contract for advisors. It allows participants to deposit a specified ERC20 token and then withdraw specified ERC20 token in phases, adhering to a lock-up period and withdrawal restrictions.
- *LakeVestingInvestors* – The contract facilitates a vesting mechanism for early investors where they deposit tokens and can later withdraw them based on certain conditions.
- *LakeVestingPrivate* – The contract is a token vesting mechanism that allows users to deposit a specified token and, in return, they can claim and withdraw the token as rewards over time. This contract allows users to lock tokens for 3 phases of three hundred sixty days each (3 years). They are also allowed to claim rewards based on the deposited amount.
- *LakeVestingTeam* - The contract facilitates a vesting mechanism for the team where they deposit tokens and can later unlock them based on certain conditions.
- *TeamMultiSig* - A multi-signature wallet used by the project's team to call some restricted functions. The initial number of owners is five, which is also the maximum number of owners. The minimum number of owners is four.

Privileged roles

- Owner: The only one who can call the control and management functions, such as transferring the ownership of the token contract, minting new tokens, or changing the receiving address of newly minted tokens.
- MultiSig Owner: All those wallet addresses englobed by the MultiSig wallet and that participate in its consensus mechanism.

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 scoring methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**.

- Technical description is robust.
- Functional requirements are detailed.

Code quality

The total Code Quality score is **9** out of **10**.

- Solidity Style Guide violations. (I02)
- Redundant interface. (I07)

Test coverage

Code coverage of the project is **100%** (branch coverage).

Security score

As a result of the audit, the code contains **no** issues. 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**. The system users should acknowledge all the risks summed up in the risks section of the report.

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
08 September 2023	5	4	4	0
28 September 2023	0	0	0	0

Risks

- Proposals lack a designated time limit or a timestamp indicating their creation. This absence of temporal context presents a considerable security risk, potentially resulting in erroneous

decision-making and the execution of transactions based on obsolete or unreliable data.

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

Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items considered:

Item	Description	Status	Related Issues
Default Visibility	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed	
Integer Overflow and Underflow	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed	
Outdated Compiler Version	It is recommended to use a recent version of the Solidity compiler.	Passed	
Floating Pragma	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed	
Unchecked Call Return Value	The return value of a message call should be checked.	Passed	
Access Control & Authorization	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	The contract should not be self-destructible while it has funds belonging to users.	Passed	
Check-Effect-Interaction	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed	
Assert Violation	Properly functioning code should never reach a failing assert statement.	Passed	
Deprecated Solidity Functions	Deprecated built-in functions should never be used.	Passed	
Delegatecall to Untrusted Callee	Delegatecalls should only be allowed to trusted addresses.	Not Relevant	
DoS (Denial of Service)	Execution of the code should never be blocked by a specific contract state unless required.	Passed	

Race Conditions	Race Conditions and Transactions Order Dependency should not be possible.	Passed	
Authorization through tx.origin	tx.origin should not be used for authorization.	Passed	
Block values as a proxy for time	Block numbers should not be used for time calculations.	Passed	
Signature Unique Id	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery. EIP-712 should be followed during a signer verification.	Not Relevant	
Shadowing State Variable	State variables should not be shadowed.	Passed	
Weak Sources of Randomness	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant	
Incorrect Inheritance Order	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	All external calls should be performed only to trusted addresses.	Passed	
Presence of Unused Variables	The code should not contain unused variables if this is not justified by design.	Passed	
EIP Standards Violation	EIP standards should not be violated.	Passed	
Assets Integrity	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed	
User Balances Manipulation	Contract owners or any other third party should not be able to access funds belonging to users.	Passed	
Data Consistency	Smart contract data should be consistent all over the data flow.	Passed	

Flashloan Attack	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	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	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	Style guides and best practices should be followed.	Failed	I02
Requirements Compliance	The code should be compliant with the requirements provided by the Customer.	Passed	
Environment Consistency	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed	
Secure Oracles Usage	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant	
Tests Coverage	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed	
Stable Imports	The code should not reference draft contracts, which may be changed in the future.	Passed	

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

Findings

■■■■ Critical

No critical severity issues were found.

■■■ High

H01. Undocumented Functionality; Redundant Functionality

Impact	Medium
Likelihood	High

Proper documentation is vital for ensuring that developers, auditors, and users understand how to interact with the contract correctly and avoid potential vulnerabilities or misuse. To address this issue, a comprehensive documentation should be developed for all non-standard functionalities.

LAKE_Token and *Team_MultiSig* contracts contain undocumented *receive()* functions. These functions are not needed for the purpose defined in documentation. Additionally, these contracts do not need to receive native tokens.

Paths: ./LAKE_Token.sol : receive();

./Team_MultiSig.sol : receive();

Recommendation: Provide purpose and documentation of *receive()* functions in mentioned contracts or delete unneeded functionality.

Found in: 11c5909

Status: Fixed (Revised commit: 56eee43)

H02. Race Condition; Unauthorized Access To Critical Functions

Impact	Medium
Likelihood	High

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

A race condition occurs when multiple transactions or processes attempt to access and modify shared data concurrently, leading to unpredictable and potentially erroneous outcomes. Race conditions can arise when multiple transactions interact with the same contract's state simultaneously. This can result in unintended state changes, data corruption, or inconsistent behavior, compromising the reliability and security of the contract.

LAKE_Vesting_Team contract is designed only to be used by team members, but the contract does not contain any access control mechanism to ensure that it will be used only by the team. Missing validation causes that *lock()* function to be called by anyone.

Lack of access control mechanism and design allows the attacker to front run *lock()* function sent by the team and lock further locks, because *lock()* function can be called only once (*depositDone* is set to true).

Path: ./LAKE_Vesting_Team.sol : lock();

Recommendation: To avert the risks associated with unauthorized access to critical functionality, it is imperative to establish a robust access control mechanism that meticulously governs access to these functions.

Found in: 11c5909

Status: Fixed (Revised commit: 56eee43)

H03. Funds Lock

Impact	Medium
Likelihood	High

Users can invoke the *lock()* method without verifying the sufficiency of tokens inside the vesting contracts. If the contract does not hold adequate tokens to satisfy the user's rewards, it can result in the user's funds becoming lock within the contract.

Paths: ./LAKE_Vesting_Advisors.sol : lock();

./LAKE_Vesting_Investors.sol : lock();

./LAKE_Vesting_Private.sol : lock();

Recommendation: Before executing the *lock()* method, introduce a validation step to ensure that the vesting contract has sufficient funds to cater to the user's rewards.

Found in: 11c5909

Status: Fixed (Revised commit: 56eee43)

H04. Denial of Service

Impact	Medium
Likelihood	High

The occurrence of a Denial of Service (DoS) can arise either through deliberate attacks or due to mishandled edge cases. This vulnerability is rooted in its ability to disrupt the operational integrity of a contract, leading to potential short-term or even enduring incapacitation. In effect, the functionalities become inaccessible to genuine users.

Team_MultiSig contract can have only one pending transaction under voting. This can lead to possible DoS when a malicious owner will front-run a desired proposal and lock the ability to submit a new proposal(for example, deleting a malicious owner).

Path: ./Team_MultiSig.sol : submitTransaction();

Recommendation: Remove checks that disable submitting proposals when there is another pending proposal under vote.

Found in: 11c5909

Status: Fixed (Revised commit: 56eee43)

Medium

M01. Checks-Effects-Interactions Pattern Violation

Impact	Medium
Likelihood	Medium

State variables are updated after the external calls to the token contract.

As explained in Solidity Security Considerations, it is best practice to follow the checks-effects-interactions pattern when interacting with external contracts to avoid reentrancy-related issues.

Paths: ./LAKE_Vesting_Advisors.sol : lock();

./LAKE_Vesting_Investors.sol : lock();

./LAKE_Vesting_Private.sol : lock();

./LAKE_Vesting_Team.sol : lock();

Recommendation: Follow the checks-effects-interactions pattern when interacting with external contracts.

Found in: 11c5909

Status: Fixed (Revised commit: 56eee43)

M02. Usage of Built-in Transfer

Impact	Medium
Likelihood	Medium

It is considered following best practices to avoid unclear situations and prevent common attack vectors.

The built-in *transfer()* and *send()* functions process a hard-coded amount of Gas. In case if the receiver is a contract with receive or fallback function, the transfer may fail due to the “out of Gas” exception.

This may lead to denial of service situations for specific accounts.

Paths: ./LAKE_Vesting_Team.sol : receive();

./Team_MultiSig.sol : receive();

Recommendation: Follow common best practices, replace *transfer()* functions with *call()*.

Found in: 11c5909

Status: Fixed (Revised commit: 56eee43)

M03. Inefficient Gas Modeling

Impact	Medium
Likelihood	Medium

The *lock()* function showcases a substantial consumption of Gas during its execution. Tokens are firstly transferred from *msg.sender* to the*LAKE_Vesting_Advisors* contract and later from contract to *BURN_ADDRESS*.

This leads to high transaction costs.

Paths: ./LAKE_Vesting_Advisors.sol : lock();

./LAKE_Vesting_Investors.sol : lock();

./LAKE_Vesting_Private.sol : lock();

Recommendation: The same outcome can be achieved with much lower Gas consumption, by transferring ERC-20 tokens from msg.sender directly to BURN_ADDRESS.

SafeERC20.safeTransferFrom(IERC20(DEPOSIT_TOKEN_ADDRESS), msg.sender, BURN_ADDRESS, _amount);.

Found in: 11c5909

Status: Fixed (Revised commit: 56eee43)

M04. Missing Event for Critical Value Updation

Impact	Medium
Likelihood	Medium

Critical state changes should emit events for tracking things off-chain.

This may lead to inability for users to subscribe events and check what is going on with the project.

Paths: ./LAKE_Token.sol : changeReceiver()

Recommendation: Emit events on critical state changes.

Found in: 11c5909

Status: Fixed (Revised commit: 56eee43)

Low

L01. Missing Zero Address Validation

Impact	Low
Likelihood	Low

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

This can lead to unwanted external calls to 0x0.

Paths: ./LAKE_Vesting_Team.sol : constructor();

./LAKE_Token.sol : changeReceiver();

./LAKE_Vesting_Advisors.sol: constructor();

./LAKE_Vesting_Investors.sol: constructor();

./LAKE_Vesting_Private.sol : constructor();

Recommendation: Implement zero address checks.

Found in: 11c5909

Status: Fixed (Revised commit: 56eee43)

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

L02. Documentation Contradiction

Impact	Low
Likelihood	Low

According to the documentation:

The admin will transfer the ownership of the contract to the MultiSig wallet address by calling the function `transferOwnership()` imported from Ownable.

To ensure that this ownership transfer will happen, it can be performed inside the *`constructor()`* of *`LAKE_Token`* contract.

Path: ./LAKE_Token.sol : constructor();

Recommendation: Add *`transferOwnership()`* inside *`constructor()`* of *`LAKE_Token`*.

Found in: 11c5909

Status: Fixed (Revised commit: 56eee43)

L03. Unfinalized Code

Impact	Low
Likelihood	Low

The provided code should be implemented in the full logic of the project. Since any missing parts, TODOs, or drafts can change in time, the robustness of the audit cannot be guaranteed.

Incomplete code impacts on project reliability and makes it harder to evaluate project security.

Paths: ./LAKE_Vesting_Advisors.sol : getUnlockableAmount(), getTimeUntilNextUnlock();

./LAKE_Vesting_Investors.sol : getUnlockableAmount(), getTimeUntilNextUnlock();

./LAKE_Vesting_Private.sol : getGeneratedRewards(), getUnlockableAmount(), getTimeUntilNextUnlock();

./LAKE_Vesting_Team.sol : getUnlockableAmount(), getTimeUntilNextUnlock();

Recommendation: It is imperative to diligently finalize the codebase in its entirety. Remove any TODO comments and commented code parts that indicate incomplete sections and ensure that all functions, components, and modules are implemented according to the project's functional requirements. By doing so, it will fortify the robustness of the code and significantly enhance its security posture.

Found in: 11c5909

Status: Fixed (Revised commit: 56eee43)

L04. Documentation Contradiction

Impact	Low
Likelihood	Low

According to the NatSpec of LAKE_Vesting_Team:

> *Each user goes through 2 phases divided…*

However, according to the implementation, only one user can lock tokens inside the contract.

Path: ./LAKE_Vesting_Team.sol : *;

Recommendation: Fix a mismatch.

Found in: 11c5909

Status: Fixed (Revised commit: 56eee43)

L05. NatSpec Contradiction

Impact	Low
Likelihood	Low

According to the NatSpec:

> *Do NOT allow to submit a new transaction if the last one is pending (not executed and not rejected)*

The NatSpec documentation instructs not to allow the submission of a new transaction if the previous one is still pending, which is defined as neither executed nor rejected. The contract's logic contradicts this by using an OR (||) operator in the require statement:

```
"require(prevTx.rejected || prevTx.executed, "Pending TX");"
```

This means that if either condition is met, the function will proceed, not adhering to the NatSpec's specifications.

Path: ./Team_MultiSig.sol : submitTransaction()

Recommendation: Fix the mismatch between the code and the Natspec.

Found in: 11c5909

Status: Fixed (Revised commit: 56eee43)

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

Informational

I01. Floating Pragma

The project uses floating pragmas >=0.8.0.

This may result in the contracts being deployed using the wrong pragma version, which is different from the one they were tested with. For example, they might be deployed using an outdated pragma version, which may include bugs that affect the system negatively.

Path: ./contracts : *,

Recommendation: Lock the pragma version whenever possible and avoid using a floating pragma in the final deployment. Consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.

Found in: 11c5909

Status: Fixed (Revised commit: 56eee43)

I02. Style Guide Violation - Naming Conventions

Solidity Style Guide Naming Conventions are violated.

According to Solidity Style Guide, contracts and libraries should be named using the CapWords style.

Paths: ./LAKE_Token.sol : contract name;

./LAKE_Vesting_Advisors.sol : contract name;

./LAKE_Vesting_Investors.sol : contract name;

./LAKE_Vesting_Private.sol : contract name;

./LAKE_Vesting_Team.sol : contract name;

./Team_MultiSig.sol : contract name;

Recommendation: Consistent adherence to the official Solidity style guide is recommended. This enhances readability and maintainability of the code, facilitating seamless interaction with the contracts. Change contract names to CapWords style.

Found in: 11c5909

Status: Reported (Mismatch between contract names and file names violates Solidity Style Guide. Change file names to corresponding contract names inside.)

I03. Inefficient Gas Modeling

Time related variables can be down casted from *uint256* to *uint64*.

That will lower transaction costs.

Paths: ./LAKE_Vesting_Advisors.sol : lockingTime;

./LAKE_Vesting_Investors.sol : lockingTime;

./LAKE_Vesting_Private.sol : lockingTime;

./LAKE_Vesting_Team.sol : lockingTime;

Recommendation: Down cast mentioned variables to *uint64* type.

Found in: 11c5909

Status: Fixed (Revised commit: 56eee43)

I04. Inefficient Gas Modeling

Max number of possible wallets used in Team_MultiSig contract is 5. Variables holding values related to voting can be down casted from *uint256* to *uint8* (max possible value is 5).

That will lower transaction costs.

Path: ./Team_MultiSig.sol : numConfirmationsRequired, Transaction.numConfirmations, Transaction.numRejections;

Recommendation: Down cast mentioned variables to *uint8* type.

Found in: 11c5909

Status: Fixed (Revised commit: 56eee43)

I05. Use of Hard-Coded Values

There is a hardcoded '1' in the calculation in the following line of code:

(block.timestamp - lockingTime[_wallet] - FULL_LOCKUP_PERIOD) / TIME_1_MONTH_ADJUSTED + 1;

It is unclear what the '1' represents and why it is added, making the code less readable and maintainable.

Paths: ./LAKE_Vesting_Investors.sol: getNumUnlockPeriods()

./LAKE_Vesting_Advisors.sol: getNumUnlockPeriods()

./LAKE_Vesting_Team.sol: getNumUnlockPeriods()

Recommendation: Either explain the use of these hardcoded values in the NatSpec or convert these variables into constants.

Found in: 11c5909

Status: Fixed (Revised commit: 56eee43)

I06. State Variables Default Visibility

Variable *depositDone* visibility is not specified. Specifying state variables visibility helps to catch incorrect assumptions about who can access the variable.

This makes the contract`s code quality and readability higher.

Path: ./LAKE_Vesting_Team.sol: depositDone

Recommendation: Specify variables as public, internal, or private. Explicitly define visibility for all state variables.

Found in: 11c5909

Status: Fixed (Revised commit: 56eee43)

I07. Redundant Interface

Impact	Low
Likelihood	Low

The interface *Token* is declared in *LakeVestingTeam* but never used.

The interface *Token* is declared in *LakeVestingPrivate* but *IERC20* imported in *SafeERC20* can be used instead.

The interface *Token* is declared in *LakeVestingInvestors* but *IERC20* imported in *SafeERC20* can be used instead.

The interface *Token* is declared in *LakeVestingAdvisors* but *IERC20* imported in *SafeERC20* can be used instead.

This redundancy in import operations has the potential to result in unnecessary gas consumption during deployment and could potentially impact the overall code quality.

Paths: ./LAKE_Vesting_Advisors.sol : Token;

./LAKE_Vesting_Investors.sol : Token;

./LAKE_Vesting_Private.sol : Token;

./LAKE_Vesting_Team.sol : Token;

Recommendation: Remove redundant *Token* interface and use *IERC20* interface imported in *SafeERC20*. Ensure that the contract is imported only in the required locations, avoiding unnecessary duplications.

Found in: 56eee43

Status: New

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 based on best industry practices at the time of the writing 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 report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and 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.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

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 Consultant 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

Appendix 1. Severity Definitions

When auditing smart contracts Hacken is using a risk-based approach that considers the potential impact of any vulnerabilities and the likelihood of them being exploited. The matrix of impact and likelihood is a commonly used tool in risk management to help assess and prioritize risks.

The impact of a vulnerability refers to the potential harm that could result if it were to be exploited. For smart contracts, this could include the loss of funds or assets, unauthorized access or control, or reputational damage.

The likelihood of a vulnerability being exploited is determined by considering the likelihood of an attack occurring, the level of skill or resources required to exploit the vulnerability, and the presence of any mitigating controls that could reduce the likelihood of exploitation.

Risk Level	High Impact	Medium Impact	Low Impact
High Likelihood	**Critical**	**High**	**Medium**
Medium Likelihood	**High**	**Medium**	**Low**
Low Likelihood	**Medium**	**Low**	**Low**

Risk Levels

Critical: Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.

High: High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.

Medium: Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.

Low: Major deviations from best practices or major Gas inefficiency. These issues won't have a significant impact on code execution, don’t affect security score but can affect code quality score.

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

Impact Levels

High Impact: Risks that have a high impact are associated with financial losses, reputational damage, or major alterations to contract state. High impact issues typically involve invalid calculations, denial of service, token supply manipulation, and data consistency, but are not limited to those categories.

Medium Impact: Risks that have a medium impact could result in financial losses, reputational damage, or minor contract state manipulation. These risks can also be associated with undocumented behavior or violations of requirements.

Low Impact: Risks that have a low impact cannot lead to financial losses or state manipulation. These risks are typically related to unscalable functionality, contradictions, inconsistent data, or major violations of best practices.

Likelihood Levels

High Likelihood: Risks that have a high likelihood are those that are expected to occur frequently or are very likely to occur. These risks could be the result of known vulnerabilities or weaknesses in the contract, or could be the result of external factors such as attacks or exploits targeting similar contracts.

Medium Likelihood: Risks that have a medium likelihood are those that are possible but not as likely to occur as those in the high likelihood category. These risks could be the result of less severe vulnerabilities or weaknesses in the contract, or could be the result of less targeted attacks or exploits.

Low Likelihood: Risks that have a low likelihood are those that are unlikely to occur, but still possible. These risks could be the result of very specific or complex vulnerabilities or weaknesses in the contract, or could be the result of highly targeted attacks or exploits.

Informational

Informational issues are mostly connected to violations of best practices, typos in code, violations of code style, and dead or redundant code.

Informational issues are not affecting the score, but addressing them will be beneficial for the project.

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

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Initial review scope

Repository	https://github.com/lakecompany/LAKE_SCs
Commit	11c59092241ac0c161c11059e00d614c60558dfd
Whitepaper	https://files.lak3.io/LAKE_Whitepaper.pdf
Requirements	https://github.com/lakecompany/LAKE_SCs/blob/main/docs/Documentation.pdf
Technical Requirements	https://github.com/lakecompany/LAKE_SCs/blob/main/README.txt
Contracts	File: contracts/LAKE - Token Contract/LAKE_Token.sol SHA3: 3a65eeb0b341097b01ac50a4d893d0815b2bcb9fd6375774c4b13c41e9fe1c7c File: contracts/LAKE - Vesting Contracts/LAKE_Vesting_Advisors.sol SHA3: 50527aca456b04414b017dbd6de75cd35e3df9bc96a90197a8487502b4c95c9d File: contracts/LAKE - Vesting Contracts/LAKE_Vesting_Investors.sol SHA3: 3b0e78241bd359eecac22930907547bafb326aa6f3b2c6f4cceb8658b5305166 File: contracts/LAKE - Vesting Contracts/LAKE_Vesting_Private.sol SHA3: cbb5aea0e377448cc1c66f79c7b0d6ddcb527cb62614c706b490a18ab1c3be03 File: contracts/LAKE - Vesting Contracts/LAKE_Vesting_Team.sol SHA3: 1c9f1a9f37b8098f61ef34b8d1d125324621648cfbdb023af9366c3944e23773 File: contracts/MultiSig/Team_MultiSig.sol SHA3: 31314502b6a32faa3439620dded63df3bfa697707c8b0c9e4ea198df45758754

Second review scope

Repository	https://github.com/lakecompany/LAKE_SCs
Commit	56eee439777b6dd163989a1d729d47d02bf222aa
Whitepaper	https://files.lak3.io/LAKE_Whitepaper.pdf
Requirements	https://github.com/lakecompany/LAKE_SCs/blob/main/docs/Documentation.pdf
Technical Requirements	https://github.com/lakecompany/LAKE_SCs/blob/main/README.txt
Contracts	File: contracts/LAKE - Token Contract/LAKE_Token.sol SHA3: 49c8a7635ca4ca732ff875c1094aa7e35a4484d7898373b29fc12c2d6480de9c File: contracts/LAKE - Vesting Contracts/LAKE_Vesting_Advisors.sol SHA3: 29ce300235c5bb25a3d2cdfeefca36411028343c292994e8153da392e5830dc9

	File: contracts/LAKE - Vesting Contracts/LAKE_Vesting_Investors.sol SHA3: de220f6188db31553b327952b00da2b575e42dd96730b63f4af7b8a4807e9df0 File: contracts/LAKE - Vesting Contracts/LAKE_Vesting_Private.sol SHA3: 5967ada5b1fa59eca34ffd2a9b9544abb72bc36be9023bf6b40d5dd96765aed9 File: contracts/LAKE - Vesting Contracts/LAKE_Vesting_Team.sol SHA3: e35a5fd92a430eefb8b589acdab0a208963695115a34db57a23f654420764463 File: contracts/MultiSig/Team_MultiSig.sol SHA3: d5d38e719c3c76976b2e7ed1d520fa7006be133b59d32011a9859f287396913d