HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: LitLab Games
Date: May 5, 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 LitLab Games
Approved By	Marcin Ugarenko | Lead Solidity SC Auditor at Hacken OU
Type	ERC20 token; Staking; Vesting; Gaming;
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://litlabgames.com/
Changelog	21.03.2023 - Initial Review 13.04.2023 - Second Review 05.05.2023 - 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

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

Introduction

Hacken OÜ (Consultant) was contracted by LitLab Games (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 includes the following smart contracts from the provided repository:

Initial review scope

Repository	https://github.com/jgomes79/LitLabGames/
Commit	1b7b59ccdb29c3d95ebdb9080819abbb707a93ba
Whitepaper	https://litlabgames.com/Whitepaper.pdf
Functional Requirements	https://docs.google.com/spreadsheets/d/1RRhlJmAlpxNtDiE5yHsmWqggSeV3NXw3/edit#gid=1583403091
Technical Requirements	https://litlabgames.com/Whitepaper.pdf
Contracts	File: ./contracts/game/CyberTitansGame.sol SHA3: 8032895ea8f8d411f55f6714bb834f153297b6f287fb4b006dd35693c1bd58bf File: ./contracts/game/CyberTitansTournament.sol SHA3: bb4ec5e67021da96bb20c24ae46d3eeaf15341698f6b69d58bb71160d45e16a8 File: ./contracts/metatx/LitlabContext.sol SHA3: c313cdcf186261a8ab45420b272a1780740f127c99abdf303ce2cae50e55f4d0 File: ./contracts/metatx/LitlabForwarder.sol SHA3: 97a18cb51e98bec06c7659fe8e2a1752f7e9b11c715d3706de02eb453ccf5539 File: ./contracts/staking/LitlabPreStakingBox.sol SHA3: 47d64cbad8a1d18c0814fa50a2e4ba03099318a1108f24ea033ad950b3a4cbe2 File: ./contracts/token/ILitlabGamesToken.sol SHA3: 45a745f0068b21916d0d0576d409d5cbb6c69150ce013c41d0e8ca2a42d95f20 File: ./contracts/token/LitlabGamesToken.sol SHA3: 2132366d7464aa235dc87b375402f45724059c0b1f5a9f39d492a717e04098bc File: ./contracts/utils/Ownable.sol SHA3: 833038cf88fddc1119f0d74b0063bbfd2655234af618db626ad1354a9dde0342 File: ./contracts/vesting/LITTAdvisorsTeam.sol SHA3: 87d0bccd8e5a0c6912a00d977404800310e93af7eebb6fcdb00d89ce7cf94fac File: ./contracts/vesting/LITTVestingContract.sol SHA3: 7f37499d7a04276396f356b3323ada88637b07b3752db4b5bb060b9dc24e7946

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

Second review scope

Repository	https://github.com/jgomes79/LitLabGames/
Commit	454e3b0f7aa8150a93a19c85d8d7e40fcaa052e8
Whitepaper	https://litlabgames.com/Whitepaper.pdf
Functional Requirements	https://docs.google.com/spreadsheets/d/1RRhlJmAlpxNtDiE5yHsmWqggSeV3NXw3/edit#gid=1583403091
Technical Requirements	https://litlabgames.com/Whitepaper.pdf
Contracts	File: ./smartcontracts/contracts/game/CyberTitansGame.sol SHA3: 3e676530b560919f39f6c7a3a7f3368c10719d891c8b8f0f67e51c7da97a4dba File: ./smartcontracts/contracts/game/CyberTitansTournament.sol SHA3: 0f52e05343b49f99ad43cc4a00c84ec524a162e4a9775e384961599f6600e9e9 File: ./smartcontracts/contracts/metatx/LitlabContext.sol SHA3: c313cdcf186261a8ab45420b272a1780740f127c99abdf303ce2cae50e55f4d0 File: ./smartcontracts/contracts/metatx/LitlabForwarder.sol SHA3: 97a18cb51e98bec06c7659fe8e2a1752f7e9b11c715d3706de02eb453ccf5539 File: ./smartcontracts/contracts/staking/LitlabPreStakingBox.sol SHA3: 99d06dc8ecb68f015c22f3ef8c42426b3ddaebbc52f05ab3fdfaa3945fa50681 File: ./smartcontracts/contracts/token/ILitlabGamesToken.sol SHA3: 45a745f0068b21916d0d0576d409d5cbb6c69150ce013c41d0e8ca2a42d95f20 File: ./smartcontracts/contracts/token/LitlabGamesToken.sol SHA3: be5ba8d86fc1dbb001cf3da7e54d0b722a42f42c2a53cc4c29dd91f7a4493025 File: ./smartcontracts/contracts/utils/Ownable.sol SHA3: 65d3bb6971cd315b2dea7558920386795287394e5d0c177b2600fbf94e019e78 File: ./smartcontracts/contracts/vesting/LITTAdvisorsTeam.sol SHA3: 12a7116c5b6b08f6a478badca8b74aa667a7c4d80100caa35c2c32385e1d0adb File: ./smartcontracts/contracts/vesting/LITTVestingContract.sol SHA3: 81488c37b5bb2b0f0b5b5880cdbdd2f00b342c3c74bdf1e5b1edd2eafdc938e2

Third review scope

Repository	https://github.com/jgomes79/LitLabGames/
Commit	ab293ecaf7ed754f964b7c7b5e31985883db4504
Whitepaper	https://litlabgames.com/Whitepaper.pdf
Functional Requirements	https://docs.google.com/spreadsheets/d/1RRhlJmAlpxNtDiE5yHsmWqggSeV3NXw3/edit#gid=1583403091
Technical Requirements	https://litlabgames.com/Whitepaper.pdf

Contracts	File: ./contracts/game/CyberTitansGame.sol SHA3: 8caa12e77012578ee57d9e59d102aa76badd4729e2b102c8353d7e46db633a1f File: ./contracts/game/CyberTitansTournament.sol SHA3: 45a1caf1add204ea0e0615500e59c12ca3a6913d1644105abb5e96ae99ee5082 File: ./contracts/metatx/LitlabContext.sol SHA3: c313cdcf186261a8ab45420b272a1780740f127c99abdf303ce2cae50e55f4d0 File: ./contracts/metatx/LitlabForwarder.sol SHA3: 97a18cb51e98bec06c7659fe8e2a1752f7e9b11c715d3706de02eb453ccf5539 File: ./contracts/staking/LitlabPreStakingBox.sol SHA3: 08aeab5b7c18fc313c98deaa5d2cb7dc40670069b4c1d4f25075e855b1bdb273 File: ./contracts/token/ILitlabGamesToken.sol SHA3: 45a745f0068b21916d0d0576d409d5cbb6c69150ce013c41d0e8ca2a42d95f20 File: ./contracts/token/LitlabGamesToken.sol SHA3: b99f916e641f5c8b9548ad681f37435aa86fa2eec18060e12f66654abcbf22d2 File: ./contracts/utils/Ownable.sol SHA3: 65d3bb6971cd315b2dea755892038679528739 4e5d0c177b2600fbf94e019e78 File: ./contracts/vesting/LITTAdvisorsTeam.sol SHA3: 77572b5cdd4ac4bf87eabefb03fba02ea4a0ddc3a1adbea8ca6fe9c592d2f521 File: ./contracts/vesting/LITTVestingContract.sol SHA3: bf79347cb39da1f7ba100a75d7f39e05b17d3c54f497e12d08de205b586ba457

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 the loss of user funds or contract state manipulation by external or internal actors.
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 by external or internal actors.
Medium	Medium vulnerabilities are usually limited to state manipulations but cannot lead to asset loss. Major deviations from best practices are also in this category.
Low	Low vulnerabilities are related to outdated and unused code or minor Gas optimization. These issues won't have a significant impact on code execution but affect code quality

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 **7** out of **10**.

- Overall system requirements are provided.
- No run instructions.
- Technical specification is provided.
- NatSpec is not fully provided.

Code quality

The total Code Quality score is **8** out of **10**.

- Development environment is not configured.

Test coverage

Code coverage of the project is **49.16%** (branch coverage).

- Test coverage is insufficient.

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: **7.2**.

The system users should acknowledge all the risks summed up in the risks section of the report.

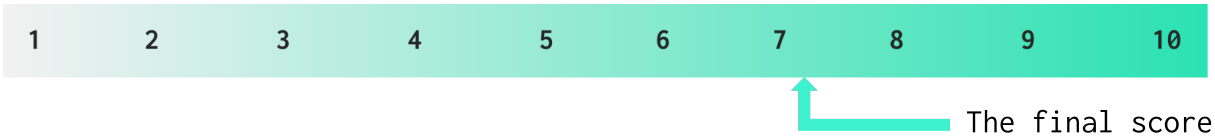

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
21 March 2023	20	20	15	5
13 April 2023	6	5	5	2
5 May 2023	0	0	0	0

Risks

- The project implements an *Antisnipe* functionality in *LitlabGamesToken* from Gotbit which cannot be validated by Hacken since it has not been provided as part of the audit scope. This functionality is meant to authorize and control transactions happening during the token generation event (TGE) by the Gotbit team.
- Both CyberTitansGame and CyberTitansTournament work in multiples of 8 players, but there is no code provided that checks and makes the groups. Instead, those player groups are imputed by the LitlabGames server off-chain and cannot be verified. This affects the functions createGame(), finalizeGame(), checkWallets(), startTournament(), finalizeTournament().
- The flow of the project is not wholly on-chain (i.e. defined in smart contracts) since the project server holds a big part of it, and hence it's not completely verifiable.
- The system is fully centralized, an owner can withdraw any available number of ERC20 tokens from the smart contracts by the use of the emergencyWithdraw() function.
- The system is accepting arbitrary tokens in the CyberTitansGame and CyberTitansTournament contracts. If those tokens are a fee-on-transfer or reflection tokens the system will not work correctly.
- After the project team responded to the issues, some of them were marked as Mitigated; those issues are not fixed, the project accepted and acknowledged those findings and took responsibility for their correctness.

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

System Overview

LitLab Games is a mixed-purpose system with the following contracts:

- *LitlabGamesToken* — ERC-20 token. With permit and burn mechanism. Extended by the external Antisnipe functionality. Total supply is minted to the deployer address.
  It has the following attributes:
  - Name: LitlabToken
  - Symbol: LITT
  - Decimals: 18
  - Total supply: 3b tokens.
- *LitlabGamesToken* — an interface for a burnable ERC20 token.
- *Ownable* - an abstract smart contract that implements the possibility of ownership.
- *LitlabPreStakingBox* - a vesting smart contract, which is used to issue an initial token offer to a limited number of users. Wheezing participants can only be added by the owner. Only the owner assigns the amount of the reward received as a result of staking. The smart contract has an option of the extra over the time rewards program for no withdrawal.
- *LITTAdvisorsTeam* - a vesting smart contract which is used to distribute tokens for advisors and the team. The smart contract uses a linear distribution of tokens. A team needs to verify three different wallets in order to withdraw their rewards.
- *LITTVestingContract* - a vesting smart contract which is used to distribute the supply of tokens in accordance with the white paper and distribution schedule.
- *CyberTitansGame* - a smart contract used for creating and managing *games*. The *game* in this case is an abstract thing, it stores a list of players and winners. Winners are chosen outside the blockchain. The contract calculates and sends the rewards for the winners.
- *CyberTitansTournament* - a smart contract used for creating and managing *tournaments*. The *tournaments*, in this case, is an abstract thing, it stores a number of participants, calculate rewards according to the amount of players. Anyone can join the tournament, winners are chosen outside the blockchain.
- *LitlabContext* - an instance of simple ERC2771Context smart contract.
- *LitlabForwarder* - a forwarder smart contract which will be used for metatransactions.

Privileged roles

- The owner of the *CyberTitansGame* contract can change addresses of manager role, address for fee collecting, native game ERC20 token address. The owner can change game winners, fees percentage,

withdrawal delay, maximum bet amount, pause/unpause smart contract and withdraw any amount of tokens from the contract.

- The manager of the *CyberTitansGame* can create and end games.
- The owner of the *CyberTitansTournament* contract can change addresses of manager role, address for fee collecting, native game ERC20 token address. The owner can change reward matrix, fees percentage, pause/unpause smart contract and withdraw any amount of tokens from the contract.
- The manager of the *CyberTitansGame* can start and end tournaments.
- The owner of the *LitlabGamesToken* can disable antisnipe system, which is not in the scope.
- The owner of the *LitlabPreStakingBox* can add new investors to staking. The owner can withdraw any amount of tokens from the contract.
- The owner of *LITTAdvisorsTeam* can set start time of vesting, add and remove advisors, set advisors' rewards, set approval wallets for team rewards withdrawing and change address of team wallet. The owner can withdraw any amount of tokens from the contract.
- The owner of *LITTVestingContract* can set the start time of vesting, change the company wallet. The owner can withdraw any amount of tokens from the contract.

Recommendations

- Owner private keys should be ⅗ multi-sig.
- Test coverage should be updated.

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	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.	Passed
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 self-destructible while 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
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Passed
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 required.	Passed

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

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.	Passed
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Passed
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155 EIP-712	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	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	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.	Passed
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 or be locked on the contract.	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

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

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
Environment Consistency	**Custom**	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Failed
Secure Oracles Usage	**Custom**	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	**Custom**	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.	Failed
Stable Imports	**Custom**	The code should not reference draft contracts, which may be changed in the future.	Passed

Findings

■■■■ Critical

C01. Data Consistency

The function *startTournament()* incorrectly compares _cttPlayers with LITTPlayers in the check:

require(_cttPlayers == tournament.numOfTokenPlayers, "BadLITTPlayers")

This will lead to incorrect data management since the checked variable is not the correct one. As a consequence, this function does not work as intended.

Path:
./contracts/game/CyberTitansTournament.sol : startTournament()

Recommendation: correct the check so that it compares _litPlayers instead of _cttPlayers.

Found in: 1b7b59c

Status: Fixed (Revised commit: 454e3b0)

C02. Invalid Calculations

In the *withdraw()* function of the *LitlabPreStakingBox* contract, the *totalRewards* value is updated incorrectly.

The *pendingReward* amount deducted from the *totalRewards* is insufficient in a case where the user is collecting their rewards using the *withdrawRewards()* function.

The already-collected user rewards should also be extracted from the *totalRewards* to not break the calculations of the *rewardsTokensPerSec* value in the *_getData()* function.

The invalid update of the *totalRewards* leads to invalid calculations of the other user rewards, resulting in too many tokens being distributed compared to the initial *totalRewards* value.

Path:
./contracts/staking/LitlabPreStakingBox.sol : withdraw()

Recommendation: track the claimed rewards amount in the *withdrawRewards()* function for each user, and extract this value from the *totalRewards* value in the case of the user's first vesting withdrawal.

Found in: 1b7b59c

Status: Fixed (Revised commit: 454e3b0)

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

C03. Invalid Calculations

In the *_executeVesting()* function, the *tokensPerSecond* value is being counted incorrectly. This is because the calculation does not take into account the portion of the tokens that are released on the TGE.

This leads to situations in which tokens are released too quickly.

Path:
./contracts/vesting/LITTVestingContract.sol : _executeVesting()

Recommendation: when calculating the *tokensPerSecond*, subtract the amount of tokens issued at TGE from the *data._amount* value. Use the same logic as in the *_calculateVestingTokens()* function.

Found in: 1b7b59c

Status: Fixed (Revised commit: 454e3b0)

C04. Data Consistency

The *setListingDate()* function can be called, at any time after the beginning of the TGE described in the documentation. Calling this function during ongoing vesting may disrupt the schedule for issuing tokens.

Path:
./contracts/vesting/LITTAdvisorsTeam.sol : setListingDate()

Recommendation: the *setListingDate()* function should only be called once or called in the constructor as an internal function.

Found in: 1b7b59c

Status: Fixed (Revised commit: ab293ec)

C05. Data Consistency

The functions *joinTournament()*, *startTournament()*, and *finalizeTournament()* take as an argument an *id* value, which is used to interact with an array of created tournaments stored in the *CyberTitansTournament* contract.

The value passed as *id* is not validated.

This issue leads to the possibility of interacting with non-existent tournaments.

For instance, a user can call the *joinTournament()* function and be able to join non-existent tournaments, tokens will be transferred from him, which essentially means their loss.

In other mentioned functions, this leads to an unexpected state of the contract data.

Path:
./contracts/game/CyberTitansTournament.sol : joinTournament(), startTournament(), finalizeTournament()

Recommendation: check if the value from the *id* parameter does not exceed the *tournamentCounter* value.

Found in: 1b7b59c

Status: Fixed (Revised commit: 454e3b0)

C06. Invalid Calculations

In the *_calculateVestingTokens()* function, the *tokensPerSecond* value is being counted incorrectly. This is because the calculation does not take into account the portion of the tokens that can be released on the TGE.

This deduction needs to be done always, not optionally, as in the line:

```
uint256 amountMinusFirstWithdraw = balances[_user].amount - (balances[_user].claimedInitial ? balances[_user].amount * INITIAL_WITHDRAW_PERCENTAGE / 100 : 0);
```

This leads to situations in which more tokens can be released than the vested amount. Users can wait the whole vesting period and call withdraw() first and withdrawInitial() afterward to extract 115% of the vested amount.

Path:
./contracts/staking/LitlabPreStakingBox.sol : _calculateVestingTokens()

Recommendation: when calculating the *tokensPerSecond* value, always subtract the *balances[_user].amount * INITIAL_WITHDRAW_PERCENTAGE / 100* from the *balances[_user].amount.*

Found in: 454e3b0

Status: Fixed (Revised commit: ab293ec)

■ ■ ■ High

H01. Invalid Hardcoded Value

According to the documentation, the values stored in the two-dimensional array *prizes[][8]* are the percentages of awards for achieving winning places in tournaments.

The values of the *prizes[5]* array, when summed up after multiplication by the number of winners per place, make a result of more than 100%.

This leads to more tokens being allocated for rewards than necessary, which can lead to an insufficient balance in the smart contract.

Path:
./contracts/game/CyberTitansTournament.sol: _buildArrays()

Recommendation: values in the *prizes[5]* array should be rounded down, for example, by replacing *164063* with *164062*.

Found in: 1b7b59c

Status: Fixed (Revised commit: 454e3b0)

H02. Insufficient Balance

In the *CyberTitansGame* contract, the *finalizeGame()* function first sends rewards to the winners, then takes fees and burns some amount of tokens.

The function does not validate the sum of these operations; it can be greater than the number of tokens allocated for the game, as there is no limit on the values used for computation: *winners[]*, *fee*, and *rake*.

This can lead to an insufficient balance in the smart contract.

Path:
./contracts/game/CyberTitansGame.sol: finalizeGame()

Recommendation: validate the amount of tokens distributed in the *finalizeGame()* function and check if it does not exceed the *totalBet* amount.

Found in: 1b7b59c

Status: Fixed (Revised commit: 454e3b0)

H03. Invalid Calculations

In the *_getData()* function, *rewardsTokensPerSec* is calculated. To do so, there are two divisions by *10**18* which are unnecessary, as they cancel each other out mathematically.

However, the way that calculation is set right now, leads to two errors:

First, Solidity language does not have floating point numbers and thus the result of the calculation will not be accurate, leaving some residual leftover tokens.

Second, the *if (totalStakedAmount > 0)* check is incorrect, as for *0 < totalStakedAmount < 1e18* range there will be a case of the division by *0*, which will result in Denial of Service violation.

Path:

./contracts/staking/LitlabPreStakingBox.sol: _getData().

Recommendation: remove the divisions by 10**18.

Found in: 1b7b59c

Status: Fixed (Revised commit: 454e3b0)

H04. Requirements Violation

In the NatSpec, it is specified that the *rake* value is used for the burn amount calculation and the *fee* for fee calculation.

In the *finalizeGame()* function, the number of tokens needed for burning is taken as fees, and the number of fees is burned (lines *141,142*).

This leads to the expected number of commissions received and tokens burned diverging from their actual values.

Path:
./contracts/game/CyberTitansGame.sol : finalizeGame()

Recommendation: change the code as per requirement.

Found in: 1b7b59c

Status: Fixed (Revised commit: 454e3b0)

H05. Undocumented Behavior

In the CyberTitansGame contract, if the token used is not $LITT, the code implementation will transfer a 5% of the game.totalBet to the team wallet instead of 2.5% burned and 2.5% sent to the team wallet.

This particular case is not reflected in the documentation. The code should not contain undocumented functionality.

Paths:
./contracts/game/CyberTitansGame.sol: finalizeGame()
./contracts/game/CyberTitansTournament.sol: finalizeTournament()

Recommendation: the provided documentation should match the code.

Found in: 1b7b59c

Status: Mitigated (with Customer notice:

"CyberTitans would be able to create games with USDCs, USDTs and other ERC20 tokens. To do that we design the createGame and createTournament functions to allow any token".)

H06. Coarse-Grained Authorization Model

The function *changeWallets()* in both CyberTitansGames.sol and CyberTitansTournament.sol sets three critical state variables at once, which can lead to dangerous situations.

A project should have a fine-grained access control system if it has multiple layers of auth-related functionality. In this case, the variable wallet is the company wallet receiving the fees; the manager has a critical access control role; and litlabToken corresponds to the token.

Additionally, this design is not efficient in terms of Gas expense, since three storage variables must be accessed every time, even if only one of them has to be set.

The code should not contain undocumented functionality.

Path:
./contracts/game/CyberTitansGame.sol: changeWallets()

Recommendation: it is recommended to use specific functions for each functionality.

Found in: 1b7b59c

Status: Fixed (Revised commit: 454e3b0)

H07. Non-Finalized Code

The function *retireFromTournament()* and the Event *onRetiredTournament* are present in the code as a draft, suggesting there will be an upgrade of the provided contracts.

This means that the code is not finalized and additional changes will be introduced in the future, which cannot be validated.

Path:
./contracts/game/CyberTitansTournament.sol: retireFromTournament(), onRetiredTournament()

Recommendation: the provided code should be final.

Found in: 1b7b59c

Status: Fixed (Revised commit: 454e3b0)

H08. Requirements Violation

Users can only join a tournament using *joinTournament()* if they make a LITT token transfer, but not if they hold the CTT tickets.

The code should match the provided documentation and intended behavior.

Path:
./contracts/game/CyberTitansTournament.sol: joinTournament().

Recommendation: the provided documentation should match the code.

Found in: 1b7b59c

Status: Mitigated (with Customer notice:

"CTT tickets are the soft currency of the game. Gamers join free games, and they can win CTT tickets to join tournaments without spending LITT tokens. The server handles in the backend the CTT tickets, so there's no interaction with SmartContracts for gamers using CTT tickets.

When starting a tournament, the server informs the SmartContract the number of players that used CTT ticket, just to add them to the players count to calculate the prizes when tournament finalizes. So, we don't need to handle users that join tournaments using CTT in the SmartContract.".)

H09. Undocumented Behavior

In order to perform a *teamWithdraw()*, a minimum *numTeamApprovals* must be reached. However, this functionality is not described in the documentation and thus a proper evaluation is not possible.

It is not clear how many approvals are needed, although the state variable *MAX_SIGNATURES_TEAM* points towards no more than three.

On the other hand, the function *setApprovalWallets()* has an input array of up to 5 wallets. If this function is used to set 5 new wallets, the state variable *numTeamApprovals* should be reset to make sure old wallets are not taken into account anymore. Otherwise, old wallets should be checked and removed from the count if they are indeed not allowed anymore.

Path:
./contracts/vesting/LITTAdvisorsTeam.sol : teamWithdraw(), setApprovalWallets(), MAX_SIGNATURES_TEAM

Recommendation: provide a clear and consistent method and documentation for a proper evaluation of this case.

Found in: 1b7b59c

Status: Fixed (Revised commit: 454e3b0)

H10. Denial of Service

The contracts LitlabPreStakingBox, LITTAdvisorsTeam and LITTVestingContract all rely on the token balance in order to perform the following critical operations:

withdrawInitial()
withdraw()
teamWithdraw()

addAdvisor()
advisorWithdraw()
withdrawNewGames()
withdrawMarketing()
withdrawLiquidReserves()
withdrawAirdrops()
withdrawInGameRewards()
withdrawFarming()

However, there is no check in those functions that makes sure that the token balance of the contract is sufficient.

The documentation provided does not clarify how the tokens are going to be transferred into the different contracts since there is no function in them to deposit tokens during the contract creation or in other functions such as *stake()*.

As a consequence, it cannot be guaranteed that the state variables tracking the token balance of the contracts have any effect at all (e.g. *totalRewards*). It is the same case for constant variables that represent the total amount of tokens (e.g. *TEAM_AMOUNT*, *LIQUID_RESERVES_AMOUNT*).

Paths:
./contracts/vesting/LITTVestingContract.sol
./contracts/vesting/LITTAdvisorsTeam.sol
./contracts/staking/LitlabPreStakingBox.sol

Recommendation: provide a clear documentation about how the tokens will be sent to the contract and/or add a deposit function to add said tokens. Add additional checks to make sure the contract balance is enough to send out the tokens in the mentioned functions.

Found in: 1b7b59c

Status: Mitigated (with Customer notice:

“- After smartcontract is deployed, we will send all the investor tokens plus the rewards amount manually to the smartcontract

- Then we will call the stake() function with the investors data.

- This is a centralized action performed by us and nobody will add more investors or tokens.”.)

H11. Requirement Violation

The function *stake()* should be called after the deployment of the contract only once.

However, there is no check that makes sure that this is the case.

Additionally, the *stake()* function can be called after the end of staking date, which leads to the fact that rewards can be received instantly.

Path:
./contracts/staking/LitlabPreStakingBox.sol : stake()

Recommendation: redesign the function flow or add additional checks to make sure this functionality matches the idea behind it. Alternatively, the function could also be set as internal and called by the constructor.

Found in: 1b7b59c

Status: Fixed (Revised commit: ab293ec)

H12. Data Consistency

In the *finalizeTournament()* function, there is no check to prevent finalizing an already finalized tournament.

This can lead to a situation where rewards are being paid multiple times for the same tournament.

Path:
./contracts/game/CyberTitansTournament.sol : finalizeTournament()

Recommendation: add a validation that already finalized tournaments cannot be finalized again.

Found in: 1b7b59c

Status: Fixed (Revised commit: 454e3b0)

H13. Data Consistency

In the *removeAdvisor()* function, the removal of the advisor's vesting is done incorrectly.

The advisor vesting amount should not be reset to the default value of 0 by using *delete advisors[_wallet];*, but should be updated to the amount that the advisor has already claimed, taken from *advisorsWithdrawn[_wallet]*.

This is needed to prevent data consistency issues and to correctly track data with the *getAdvisorData()* view function, and to prevent underflow in the *require(advisors[msg.sender] - advisorsWithdrawn[msg.sender] > 0, "NotAllowed").*

Path:
./contracts/vesting/LITTAdvisorsTeam.sol : removeAdvisor()

Recommendation: update the functionality of advisor removal in a way that will not break other functions.

Found in: 1b7b59c

Status: Fixed (Revised commit: 454e3b0)

H14. Requirements Violation

In the *withdrawNewGames()*, *withdrawInGameRewards()*, and *withdrawFarming()* functions, the vesting schedule is not in line with the tokenomics presented in the whitepaper.

This can lead to trust issues with the community and more tokens being in circulation than described in the documentation.

Path:
./contracts/vesting/LITTVestingContract.sol : withdrawNewGames(), withdrawInGameRewards(), withdrawFarming()

Recommendation: update the whitepaper/tokenomics or update the code to be in line with the vesting schedules.

Found in: 1b7b59c

Status: Mitigated (Based on new documentation:

- New Games 9 months cliff, linearly over 54 months
- Withdraw from InGame pool (not vested)
- Withdraw from Farming pool (not vested)

with Customer notice:

“New games, marketing, liquid reserves, ingame rewards, airdrops are managed in the vesting smartcontract.

Farming is managed by external providers out of the scope of this audit.)

H15. Requirements Violation

In the *_executeVesting()* function, there is an invalid validation that prevents users from claiming TGE tokens before the cliff time.

*require(block.timestamp >= listing_date + (data._cliffMonths * 30 days), "TooEarly")*;

This leads to a situation where the TGE tokens will not be claimable at the listing date.

Path:
./contracts/vesting/LITTVestingContract.sol: _executeVesting()

Recommendation: update the code to accurately reflect the desired functionality.

Found in: 1b7b59c

Status: Mitigated (with Customer notice:

"*We've written in the documentation that cliff and TGE are not compatible. If a vesting has cliff never will have TGE % and opposite.*")

H16. Non-Finalized Code

The smart contract *LITTAdvisorsTeam.sol* uses *truffle/console.sol* and *console.log()* functions inside. This means that the code is submitted in a non-final version.

Path:
./contracts/vesting/LITTAdvisorsTeam.sol : teamWithdraw()

Recommendation: remove truffle integrations from code.

Found in: 454e3b0

Status: Fixed (Revised commit: ab293ec)

H17. Data Consistency

During the *withdrawInitial()* function execution data used in rewards calculation is not updated.

Resulting in a situation where users who did not withdraw their TGE tokens are rewarded equally as those who had withdrawn.

Path:
./contracts/staking/LitlabPreStakingBox.sol : withdraw()

Recommendation: consider updating the *withdrawInitial()* function with functionality that will decrease users' participation in the reward program by the withdrawal amount, or document this behavior as intended.

Found in: 454e3b0

Status: Mitigated (Desired behavior. Users are not punished for initial withdrawal and are encouraged to withdraw the initial tokens.

with Customer notice:

"Call withdrawInitial() doesn't affect the rewards as documented. As commented in C06 issue, we've added a require in withdraw function to ensure nobody can call it before calling withdrawInitial, but, as documented, call withdrawInitial doesn't affect to the rewards.".)

Medium

M01. Missing Events for Critical Value Update

The following functions do not emit relevant events after executing the sensitive actions of setting the fundingRate, updateTime and proposalTime, and transferring the rewards.

Paths:
./contracts/game/CyberTitansGame.sol: constructor(), changeWallets(), changeWinners(), updateFees(), updateWaitMinutes(), updateMaxBetAmount(), changePause()
./contracts/game/CyberTitansTournament.sol: constructor(), changeWallets(), updateFees(), changeArrays(), changePause(), _buildArrays()
./contracts/staking/LitlabPreStakingBox.sol: constructor(), stake().
./contracts/token/LitlabGamesToken.sol: disableAntisnipe()
./contracts/vesting/LITTAdvisorsTeam.sol: constructor(), setListingDate(), addAdvisor(), removeAdvisor(), setApprovalWallets(), setTeamWallet(), approveTeamWithdraw()
./contracts/vesting/LITTVestingContract.sol: constructor(), setListingDate(), changeWallet()

Recommendation: consider emitting events after sensitive changes take place, to facilitate tracking and notify off-chain clients following the contract's activity.

Found in: 1b7b59c

Status: Fixed (Revised commit: ab293ec)

M02. Undocumented Behavior

Although the documentation refers to LitlabToken $LITT as the token to be used in the project, the following contracts allow the use of different addresses as token inputs.

The code should not contain undocumented functionality.

Path:
./contracts/game/CyberTitansGame.sol: GameStruct.token, changeWallets(), checkWallets(), createGame()

Recommendation: the provided documentation should match the code.

Status: Mitigated (with Customer notice:

"*Contracts are prepared to allow users to bet using other ERC20 tokens as USDC in future games or tournaments.*")

M03. Inefficient Gas Model: Uncontrolled Loop of Storage Interactions

The function changeArrays() performs loops of uncontrolled iterations.

Since those loops interact with storage variables, the block gas limit can be reached and the function may fail.

Additionally, this design is not efficient in terms of Gas expense, since different storage variables must be accessed every time, even if only one of them has to be set.

Path:
./contracts/game/CyberTitansTournament.sol: changeArrays()

Recommendation: divide functionality of changeArrays() into changePrizes(), changePlayers(), changeTops(), changeWinners().

Found in: 1b7b59c

Status: Mitigated (with Customer notice:

"*Understanding the gas problem, we've decided to keep the function. If we split the function in 5 functions as recommended, we need to send 5 transactions to update the prizes matrixes and we've decided is worthier to have only one function to update everything even if it's more expensive in gas cost (the matrixes will change once per one or two years, so, we don't worry about the gas cost)*".)

M04. Inefficient Gas Model: Uncontrolled Loop of Storage Interactions

The following functions perform highly expensive storage operations inside a loop, which can reach the block Gas limit and make the functions fail:

createGame()
finalizeGame()
finalizeTournament()
stake()

Paths:
./contracts/game/CyberTitansGame.sol: createGame(), finalizeGame()
./contracts/game/CyberTitansTournament.sol: finalizeTournament()
./contracts/staking/LitlabPreStakingBox.sol: stake()

Recommendation: a redesign of the mentioned functions should be implemented in order to minimize their Gas impact: use of local variables, splitting of the body function into smaller functions that are called independently, use of bonded input variables, etc. The Customer should choose the solution that works better for the project.

Found in: 1b7b59c

Status: Fixed (Revised commit: ab293ec)

M05. Inefficient Gas Model: Storage Abuse

In the finalizeTournament() function, the state variables tournament.tournamentAssuredAmount and tournament.token are accessed multiple times, consuming Gas unnecessarily.

Path:
./contracts/game/CyberTitansTournament.sol: finalizeTournament()

Recommendation: consider creating new local memory variables to save Gas.

Found in: 1b7b59c

Status: Fixed (Revised commit: 454e3b0)

M06. Inefficient Gas Model: Storage Abuse

In the finalizeGame() function, the state variables game.token and game.totalBet are accessed multiple times, consuming Gas unnecessarily.

Path:
./contracts/game/CyberTitansGame.sol: finalizeGame()

Recommendation: consider creating new local memory variables to save Gas.

Found in: 1b7b59c

Status: Fixed (Revised commit: 454e3b0)

M07. Inefficient Gas Model: Storage Abuse

In the _executeVesting() function, the state variable data._amount is accessed multiple times, consuming Gas unnecessarily.

Path:
./contracts/vesting/LITTVestingContract.sol: _executeVesting()

Recommendation: consider creating new local memory variables to save Gas.

Found in: 1b7b59c

Status: Fixed (Revised commit: 454e3b0)

M08. Inefficient Gas Model: Cache Length

In the approveTeamWithdraw() function, a for loop iterates through approvalWallets.length. The storage variable approvalWallets will be thus read at every iteration, consuming Gas unnecessarily.

Path:
./contracts/vesting/LITTAdvisorsTeam.sol: approveTeamWithdraw()

Recommendation: consider caching the array length by creating a new memory variable length = approvalWallets.length.

Found in: 1b7b59c

Status: Fixed (Revised commit: 454e3b0)

M09. Inefficient Gas Model: Cache Length

In the teamWithdraw() function, a for loop iterates through approvalWallets.length. The storage variable approvalWallets will be thus read at every iteration, consuming Gas unnecessarily.

Path:
./contracts/vesting/LITTAdvisorsTeam.sol: teamWithdraw()

Recommendation: consider caching the array length by creating a new memory variable length = approvalWallets.length.

Found in: 1b7b59c

Status: Fixed (Revised commit: 454e3b0)

M10. Inefficient Gas Model: Non-specific View Function

In the *withdrawRewards()* function, there is a call to the view function *_getData()* in order to get pendingRewards. Said function computes a lot of variables, but only one of them is used. Although it is a view function, it will spend Gas when called by a non-view function.

Path:
./contracts/staking/LitlabPreStakingBox.sol: withdrawRewards(), _getData().

Recommendation: consider using a specific function that returns pendingRewards in order to save Gas.

Found in: 1b7b59c

Status: Fixed (Revised commit: 454e3b0)

M11. Inefficient Gas Model: Non-specific View Function

In the *withdraw()* function, there is a call to the view function *_getData()* in order to get userAmount and pendingRewards. Said function computes a lot of variables, but only two of them are used. Although it is a view function, it will spend Gas when called by a non-view function.

Path:
./contracts/staking/LitlabPreStakingBox.sol: withdraw(), _getData()

Recommendation: consider using specific functions to return userAmount and pendingRewards in order to save Gas.

Found in: 1b7b59c

Status: Fixed (Revised commit: 454e3b0)

M12. Unscalable Functionality: Duplicate Code

Same checks used in several functions overwhelm code and make further development difficult.

Path:
./contracts/game/CyberTitansGame.sol : *105, 106, 129, 130*

Recommendation: add modifiers *onlyManager*, *notPaused* to check function state.

Found in: 1b7b59c

Status: Fixed (Revised commit: 454e3b0)

M13. Unscalable Functionality: Duplicate Code

Same checks used in several functions overwhelm code and make further development difficult.

Path:
./contracts/game/CyberTitansTournament.sol : *120, 121, 138, 165, 166, 178, 179*

Recommendation: add modifiers *onlyManager*, *notPaused* to check function state.

Found in: 1b7b59c

Status: Fixed (Revised commit: 454e3b0)

M14. Inconsistent Data: Rounding Error

Functions *teamWithdraw()* and *advisorWithdraw()* calculate tokens amount that can be withdrawn by a trusted user.

Division is used for the calculation of *tokensPerSecond*, the result of which is multiplied later.

Solidity language does not have floating point numbers, the result of the calculation will not be accurate.

Path:
./contracts/vesting/LITTadvisorsTeam.sol : teamWithdraw(), advisorWithdraw()

Recommendation: change the way withdrawal amounts are calculated to give the exact amount of tokens.

Found in: 1b7b59c

Status: Fixed (Revised commit: 454e3b0)

M15. Inconsistent Data: Rounding Error

The function *_executeVesting()* calculates tokens amount that can be withdrawn by a trusted user.

Division is used for the calculation of *tokensPerSecond*, the result of which is multiplied later.

Solidity language does not have floating point numbers, the result of the calculation will not be accurate.

Path:
./contracts/vesting/LITTVestingContract.sol : _executeVesting()

Recommendation: change the way withdrawal amounts are calculated to give the exact amount of tokens.

Found in: 1b7b59c

Status: Fixed (Revised commit: 454e3b0)

M16. Inconsistent Data: Rounding Error

The functions *_calculateVestingTokens()* and *_getData()* calculate tokens amount that can be withdrawn by a trusted user.

Division is used for the calculation of *tokensPerSec* and *rewardTokensPerSec*, the result of which is multiplied later.

Solidity language does not have floating point numbers, the result of the calculation will not be accurate.

Path:
./contracts/staking/LitlabPreStakingBox.sol : _getData(), _calculateVestingToken()

Recommendation: change the way withdrawal amounts are calculated to give the exact amount of tokens.

Found in: 1b7b59c

Status: Fixed (Revised commit: 454e3b0)

M17. Invalid Calculations

The reward shares that are sent to the winners, as well as the fees, are calculated as a percentage of the game.totalBet balance.

Solidity language does not have floating point numbers and thus the result of the calculation will not be accurate, leaving some residual leftover tokens.

In order to avoid that, the calculation of those percentages should remain the same, but for the very last percentage, game.totalBalance - “the sum of the previously transferred balances” should be used.

Paths:
./contracts/game/CyberTitansGame.sol: finalizeGame()
./contracts/game/CyberTitansTournament.sol: finalizeTournament()

Recommendation: it is recommended to optimize the token percentage calculation to avoid having residual tokens.

Found in: 1b7b59c

Status: Mitigated (with Customer notice stating that residual tokens are acceptable.)

M18. Contradiction: Missing Validation

According to the whitepaper:

In the *changeWinners()* function, an additional check is required to make sure that the sum of the percentage winners is <= *95%*.

In the *updateFees()* function, the sum of rake + fee should be checked to be <= *5%*.

Paths:
./contracts/game/CyberTitansGame.sol: changeWinners(), updateFees()
./contracts/game/CyberTitansTournament.sol: updateFees()

Recommendation: add recommended values validations.

Found in: 1b7b59c

Status: Fixed (Revised commit: ab293ec)

M19. Contradiction: Documentation Mismatch

According to the documentation, the array *players[]* should be of the size *8*. Therefore, additional validation should be implemented.

The require in the createGame() function *require(_players.length != 0, "BadArray")* is incorrect since it should check the array length == 8.

Path:
./contracts/game/CyberTitansGame.sol: createGame()

Recommendation: apply the mentioned checks or update the documentation accordingly.

Found in: 1b7b59c

Status: Mitigated (with Customer notice:

"*Even though in the official whitepaper we say a game is for 4/8 players, technically, the game smartcontract needs to support any number of players, so, the validation required is invalid and doesn't fit with the requirements.*".)

M20. Contradiction: Documentation Mismatch

In the *changeArrays()* function, there is no check that the provided values are correct and will not cause unexpected behavior or break the contract's logic.

Path:
./contracts/game/CyberTitansGame.sol : changeArrays()

Recommendation: implement input checks to make sure the parameters introduced in said function are as expected.

Found in: 1b7b59c

Status: Fixed (Revised commit: ab293ec)

Low

L01. Floating Pragma

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Path:
./contracts/utils/Ownable.sol

Recommendation: lock the Solidity pragma version. Find more: SWC-103.

Found in: 1b7b59c

Status: Fixed (Revised commit: 454e3b0)

L02. Style Guide: Order of Functions

The provided projects should follow the official guidelines. Functions should be grouped according to their *visibility* and ordered:

1. Constructor
2. Receive function (if exists)
3. Fallback function (if exists)
4. External
5. Public
6. Internal
7. Private

Path:
./contracts/game/CyberTitansTournament.sol

Recommendation: follow the official Solidity guidelines.

Found in: 1b7b59c

Status: Fixed (Revised commit: 454e3b0)

L03. Style Guide: Order of Layout

The provided projects should follow the official guidelines. Inside each contract, library or interface, use the following order:

1. Type declarations
2. State variables
3. Events
4. Modifiers
5. Functions

Path:
./contracts/utils/Ownable.sol

Recommendation: follow the official Solidity guidelines.

Found in: 1b7b59c

Status:Fixed (Revised commit: 454e3b0)

L04. Style Guide: Event Names

Events should be named using the *CapWords* style.

Paths:
./contracts/game/CyberTitansGame.sol
./contracts/game/CyberTitansTournament.sol
./contracts/staking/LitlabPreStakingBox.sol
./contracts/vesting/LITTAdvisorsTeam.sol
./contracts/vesting/LITTVestingContract.sol

Recommendation: follow the official Solidity guidelines.

Found in: 1b7b59c

Status: Fixed (Revised commit: 454e3b0)

L05. Recommendation: Indexed Inputs in Events

Events have the possibility to track their inputs as *indexed*. It is recommended to use the *indexed* keyword for better tracking of sensitive data.

Paths:
./contracts/game/CyberTitansGame.sol
./contracts/game/CyberTitansTournament.sol
./contracts/staking/LitlabPreStakingBox.sol
./contracts/vesting/LITTAdvisorsTeam.sol
./contracts/vesting/LITTVestingContract.sol

Recommendation: consider adding the indexed keyword to track token addresses in events.

Found in: 1b7b59c

Status: Fixed (Revised commit: ab293ec)

L06. Missing Zero Address Validation

Address input parameters are being used without checking against the possibility of *0x0*. This can lead to unwanted external calls to *0x0*.

Paths:
./contracts/game/CyberTitansGame.sol : constructor(), changeWallets(), createGame(), finalizeGame(), emergencyWithdraw()
./contracts/game/CyberTitansTournament.sol : constructor(), changeWallets(), finalizeTournament(), emergencyWithdraw()
./contracts/staking/LitlabPreStakingBox.sol : constructor(), stake(), emergencyWithdraw()
./contracts/vesting/LITTAdvisorsTeam.sol : constructor(), addAdvisor(), removeAdvisor(), setApprovalWallets(), setTeamWallet(), emergencyWithdraw()

./contracts/vesting/LITTVestingContract.sol : constructor(), changeWallet()

Recommendation: add zero address checks.

Found in: 1b7b59c

Status: Fixed (Revised commit: 454e3b0)

L07. State Variable Default Visibility

Specifying state variables' visibility helps catching incorrect assumptions about who can access the variable.

This improves the code quality and readability of the code.

Paths:
./contracts/game/CyberTitansGame.sol : gameCounter
./contracts/game/CyberTitansTournament.sol : tournamentCounter
./contracts/vesting/LITTAdvisorsTeam.sol : numTeamApprovals

Recommendation: it is recommended to specify the visibility of all state variables.

Found in: 1b7b59c

Status: Fixed (Revised commit: 454e3b0)

L09. Style Guide: Maximum Line Length

The provided projects should follow the official guidelines. Maximum suggested line length 120 is not followed.

Paths:
./contracts/game/CyberTitansGame.sol
./contracts/game/CyberTitansTournament.sol
./contracts/vesting/LITTAdvisorsTeam.sol
./contracts/vesting/LITTVestingContract.sol

Recommendation: follow the official Solidity guidelines.

Found in: 1b7b59c

Status: Fixed (Revised commit: 454e3b0)

L10. Use of Hard-Coded Values

Hard-coded values are used in computations.

Paths:
./contracts/vesting/LITTAdvisorsTeam.sol : advisorWithdraw(), teamWithdraw(), getAdvisorData()
./contracts/vesting/LITTVestingContract.sol : _executeVesting()
./contracts/token/LitlabGamesToken.sol : constructor()

Recommendation: convert these variables into constants.

Found in: 1b7b59c

Status: Fixed (Revised commit: ab293ec)

L11. Boolean Equality

Boolean constants can be used directly and do not need to be compared to *true* or *false*.

Paths:
./contracts/game/CyberTitansGame.sol : createGame(), finalizeGame()
./contracts/game/CyberTitansTournament.sol : startTournament(), finalizeTournament(), joinTournament(), createTournament()

Recommendation: remove boolean equality.

Found in: 1b7b59c

Status: Fixed (Revised commit: 454e3b0)

L12. Unused Variable

The variable *ADVISORS_AMOUNT* is never used.

Path:
./contracts/vesting/LITTAdvisorsTeam.sol

Recommendation: remove unused variable.

Found in: 1b7b59c

Status: Fixed (Revised commit: 454e3b0)

L13. Unnecessary Variable Declaration

The variables *user*, *amount* is unnecessary to function execution.

Path:
./contracts/staking/LitlabPreStakingBox.sol : stake()

Recommendation: remove unnecessary variables.

Found in: 1b7b59c

Status: Fixed (Revised commit: 454e3b0)

L14. Variables That Can Be Set as Immutable

The variables *token*, *stakeStartDate* and *stakeEndDate* are only set in the constructor and can thus be set as *immutable*.

Paths:
./contracts/staking/LitlabPreStakingBox.sol
./contracts/vesting/LITTAdvisorsTeam.sol
./contracts/vesting/LITTVestingContract.sol

Recommendation: it is recommended to set said variables as *immutable* in order to save Gas.

Found in: 1b7b59c

Status: Fixed (Revised commit: 454e3b0)

L15. Variables That Can Be Set as Constant

The variables *ADVISORS_AMOUNT*, *TEAM_AMOUNT*, *MAX_SIGNATURES_TEAM*, *NEW_GAMES_AMOUNT*, *MARKETING_AMOUNT*, *LIQUID_RESERVES_AMOUNT*, *AIRDROPS_AMOUNT*, *INGAME_REWARDS_AMOUNT* and *FARMING_AMOUNT* never change and can thus be set as *constant*.

Paths:
./contracts/vesting/LITTAdvisorsTeam.sol
./contracts/vesting/LITTVestingContract.sol

Recommendation: it is recommended to set said variables as *constant* in order to save Gas.

Found in: 1b7b59c

Status: Fixed (Revised commit: 454e3b0)

L16. Error Messages

In the *withdraw()* function, the check *require(balances[msg.sender].withdrawn < balances[msg.sender].amount, "Max")* is not sending a comprehensive error message.

In the *disableAntisnipe()* function, the check *require(!antisnipeDisable)* does not provide any error message.

Error messages are intended to notify users about failing conditions, and should provide enough information so that the appropriate corrections needed to interact with the system can be applied. Uninformative error messages greatly damage the overall user experience, thus lowering the system's quality.

If the mentioned require statement fails the checked condition, the transaction will revert silently without an informative error message.

Paths:
./contracts/staking/LitlabPreStakingBox.sol : withdraw()
./contracts/token/LitlabGamesToken.sol : disableAntisnipe()

Recommendation: appropriate error messages should be provided for both cases, that give meaningful information to the users.

Found in: 1b7b59c

Status: Fixed (Revised commit: 454e3b0)

L17. Incorrect NatSpec

The NatSpec description of the contract CyberTitansTournament refers to game instead of tournament in the line #13.

Path:
./contracts/game/CyberTitansTournament.sol

Recommendation: update the NatSpec so that it reflects the code.

Found in: 1b7b59c

Status: Fixed (Revised commit: 454e3b0)

L18. Typos

The following typos were found into the provided code:

CyberTitansGame.sol:
playes -> players

CyberTitansTournament.sol:
playes -> players
minimun -> minimum
becuase -> because
Initializate -> Initialize

LitlabPreStakingBox.sol:
splitBadLenghts -> BadLengths
splitted -> split
deployement -> deployment
witdrawRewards -> withdrawRewards

LitlabGamesToken.sol:
malicius -> malicious

Ownable.sol:
functionlity -> functionality

Paths:
./contracts/game/CyberTitansGame.sol
./contracts/game/CyberTitansTournament.sol
./contracts/staking/LitlabPreStakingBox.sol
./contracts/token/LitlabGamesToken.sol
./contracts/utils/Ownable.sol

Recommendation: correct the spelling of the mentioned typos.

Found in: 1b7b59c

Status: Fixed (Revised commit: 454e3b0)

L19. Misleading Name

The state variable winners is misleading since it stores percentages, and should be named as such.

Path:
./contracts/game/CyberTitansGame.sol

Recommendation: it is recommended to use a name that represents precisely the variable, like winnerShares or winnerPercentages.

Found in: 1b7b59c

Status: Fixed (Revised commit: 454e3b0)

L20. Best Practice Violation: Explicit Uint Size

The variables uint gameId and uint tournamentId do not explicitly set the uint size.

Paths:
./contracts/game/CyberTitansGame.sol : createGame()
./contracts/game/CyberTitansTournament : createTournament()

Recommendation: it is recommended to explicitly set the uint size as uint256 instead of using the default uint.

Found in: 1b7b59c

Status: Fixed (Revised commit: 454e3b0)

L21. Variables That Can Be Set as Immutable

The variables *token* are only set in the constructor and can thus be set as *immutable.*

Paths:
./contracts/vesting/LITTAdvisorsTeam.sol
./contracts/vesting/LITTVestingContract.sol

Recommendation: it is recommended to set said variables as *immutable* in order to save Gas.

Found in: 454e3b0

Status: Fixed (Revised commit: ab293ec)

L22. Duplicate Code

The lines *155* and *161* (*IERC20(token).safeTransfer(msg.sender, tokensToSend)*) are the same and can be replaced by just one, below the *if* statement.

Path:
./contracts/staking/LitlabPreStakingBox.sol : withdraw()

Recommendation: it is recommended to avoid code duplication in order to save Gas.

Found in: 454e3b0

Status: Fixed (Revised commit: ab293ec)

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.