Cyberscope

Audit Report

Litas

July 2025

Network	ETH
Address	0xFA63503F9e61fd59Cbea137C122Fa55c2dafF14a
Audited by	© cyberscope

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	MC	Missing Check	Unresolved
●	MSR	Missing Staking Rewards	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
Critical	Highly Likely / High Impact
Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
Minor / Informative	Unlikely / Low to no Impact

Review

Contract Name	Litas
Compiler Version	v0.8.24+commit.e11b9ed9
Optimization	200 runs
Explorer	https://etherscan.io/address/0xfa63503f9e61fd59cbea137c122fa55c2daff14a
Address	0xfa63503f9e61fd59cbea137c122fa55c2daff14a
Network	ETH
Symbol	LITAS
Decimals	18
Total Supply	100.000.000
Badge Eligibility	Yes

Audit Updates

Initial Audit	28 Feb 2025 https://github.com/cyberscope-io/audits/blob/main/litas/v1/audit.pdf
Corrected Phase 2	11 Jul 2025

Source Files

Filename	SHA256
Litas.sol	68e7c19e973900572ee77ccbdf7a30c53f4762cb3c7311a88eff5b10f6838101

Overview

Litas Token Contract

The Litas contract is an ERC20 token implementation that includes additional functionality for burning tokens and staking. It utilizes OpenZeppelin's standard implementations for ERC20 and security features such as `ReentrancyGuard` to prevent reentrancy attacks. The contract allows users to stake their tokens for a specified duration and claim them once the lock period ends.

Token Initialization

The contract is initialized with a specified name, symbol, and initial supply, which is minted to a designated owner. It follows the ERC20 standard from OpenZeppelin, ensuring compatibility with existing token infrastructure.

Token Burning

The `burn` function allows users to destroy a specified amount of their tokens, permanently reducing the total supply. When tokens are burned, the contract emits a `TokensBurned` event, logging the burner's address and the amount burned.

Staking Mechanism

The contract enables users to stake their tokens for a given duration. When a user stakes tokens, the contract transfers the tokens from the user's balance to itself and records the stake details, including the amount of tokens staked, the start timestamp of the stake, the end timestamp when the stake can be claimed, and a flag indicating whether the stake has been claimed. Stakes are stored in a mapping, allowing each user to maintain multiple active stakes. The contract emits a `TokensStaked` event upon successful staking.

Claiming Staked Tokens

After the staking period ends, users can claim their staked tokens using the `claimStakedTokens` function. This function verifies that the stake exists, ensures the lock period has expired, and checks that the stake has not already been claimed. Once verified, the contract marks the stake as claimed and transfers the tokens back to the user. A `TokensClaimed` event is emitted to record the transaction.

Retrieving Stake Details

The contract provides a getStakeDetails function, which allows users to retrieve information about a specific stake.

Roles

Users

Users (token holders and stakers) can interact with the following functions:

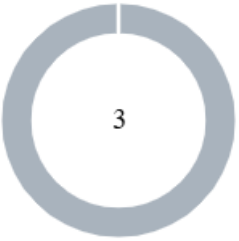
- `function burn(uint256 amount)`
- `function stake(uint256 amount, uint256 durationInDays)`
- `function claimStakedTokens(uint256 stakeIndex)`

Retrieval Functions

The following functions can be used to retrieve staking-related information:

- `function getStakeDetails(address user, uint256 stakeIndex)`

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	3

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	3	0	0	0

MC - Missing Check

Criticality	Minor / Informative
Location	Litas.sol#L752
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues.

Specifically, in the `constructor` a check is missing to ensure that `initialSupply` is not zero.

Additionally, in the `stake` function, a check is missing to ensure that the staking happens for more than zero days.

```
constructor(
    string memory name,
    string memory symbol,
    uint256 initialSupply,
    address initialOwner
)
    ERC20(name, symbol)
{
    _mint(initialOwner, initialSupply * 10**decimals());
}
function stake(uint256 amount, uint256 durationInDays) public
nonReentrant {
    require(amount > 0, "Cannot stake 0 tokens");
    require(balanceOf(msg.sender) >= amount, "Insufficient
token balance");
    //...
}
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

MSR - Missing Staking Rewards

Criticality	Minor / Informative
Location	Litas.sol#L778,802
Status	Unresolved

Description

The contract has a staking system that allows users (token owners) to stake their tokens by storing them inside the contract. Users are able to choose the amount of time (in days) they want to stake their tokens. Normally, staking contracts provide a form of reward, usually depending on the amount of time users keep their tokens staked. This incentivizes users to keep their tokens staked for longer periods of time. However, the contract does not implement any functionality that rewards users the longer they keep their tokens staked.

```
function stake(uint256 amount, uint256 durationInDays) public
nonReentrant {/*...*/}

function claimStakedTokens(uint256 stakeIndex) public
nonReentrant {
    //...
    require(block.timestamp >= stakeData.endTime, "Stake is
still locked");
    //...
    _transfer(address(this), msg.sender, stakeData.amount);
    //...
}
```

Recommendation

The team could consider adding some form of reward mechanism that accounts for the duration users keep their tokens staked. This will potentially encourage more users into staking their tokens in the protocol for longer periods of time.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	Litas.sol#L702
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.24;
```

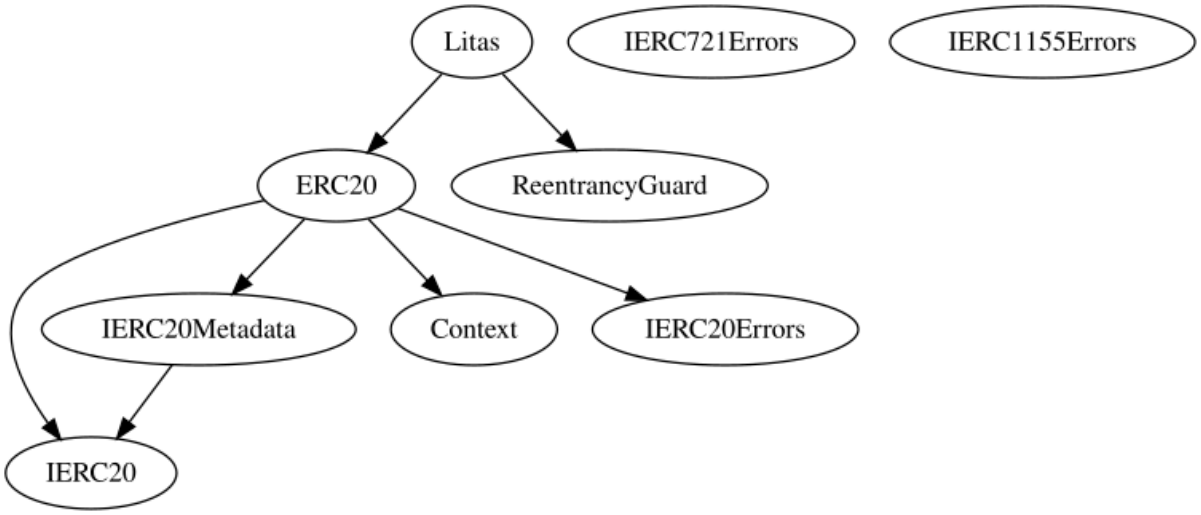
Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

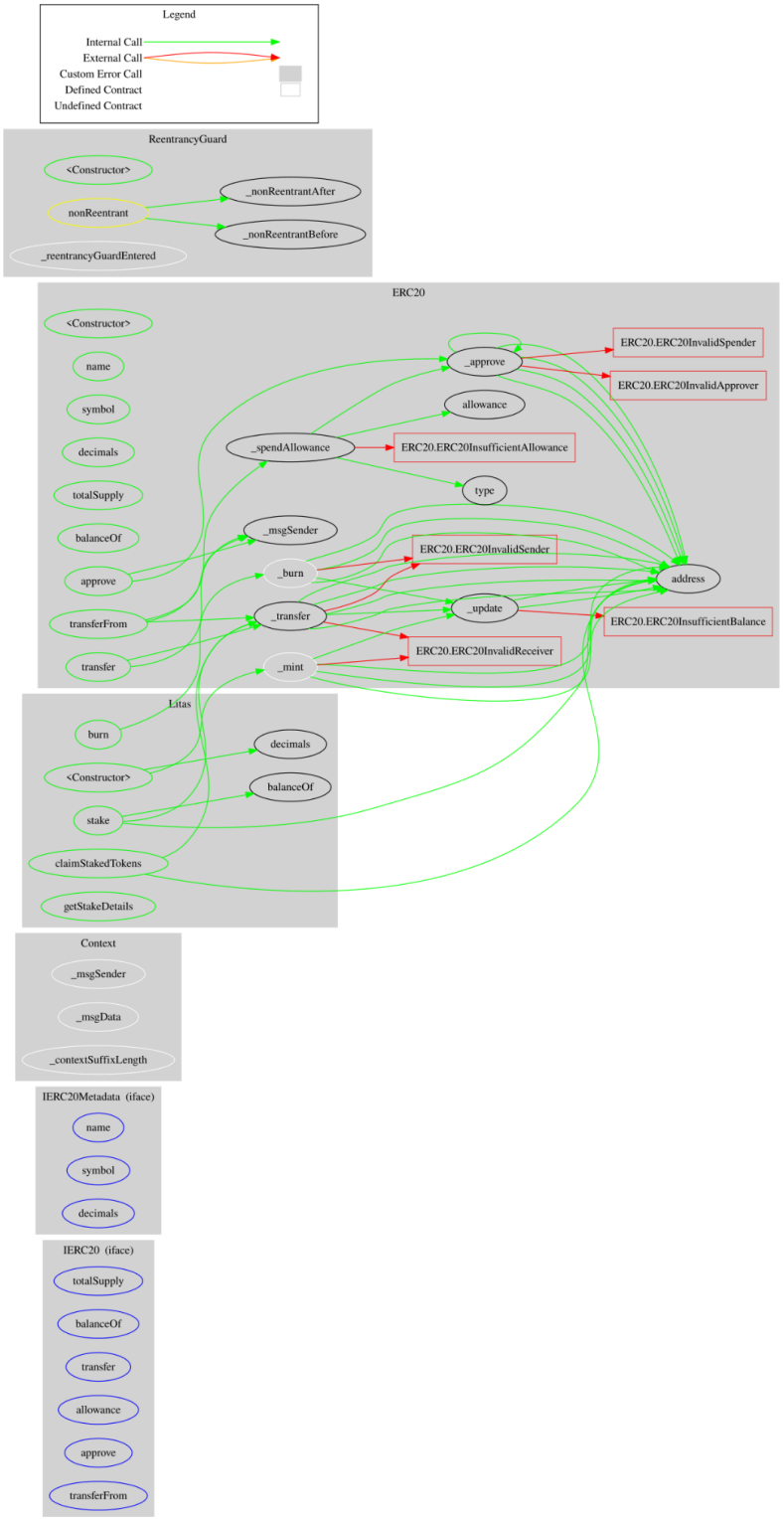
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Litas	Implementation	ERC20, ReentrancyGuard		
		Public	✓	ERC20
	burn	Public	✓	-
	stake	Public	✓	nonReentrant
	claimStakedTokens	Public	✓	nonReentrant
	getStakeDetails	Public		-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
ReentrancyGuard
<Constructor>
nonReentrant
_reentrancyGuardEntered
_nonReentrantAfter
_nonReentrantBefore
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
approve
transferFrom
transfer
_approve
_spendAllowance
allowance
type
_msgSender
_burn
_transfer
_update
_mint
address
ERC20.ERC20InvalidSpender
ERC20.ERC20InvalidApprover
ERC20.ERC20InsufficientAllowance
ERC20.ERC20InvalidSender
ERC20.ERC20InsufficientBalance
ERC20.ERC20InvalidReceiver
Litas
burn
<Constructor>
stake
claimStakedTokens
getStakeDetails
decimals
balanceOf
Context
_msgSender
_msgData
_contextSuffixLength
IERC20Metadata (iface)
name
symbol
decimals
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom

Summary

Litas contract implements a token and staking mechanism. This audit investigates security issues, business logic concerns and potential improvements. Litas is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

cyberscope.io