Cyberscope

Audit Report

Libera Global AI

November 2024

Files	LiberaContract
SHA 256	2cc954b5e79cbb686946f336f76591ef5c6f2d7217c8d07b6f80bebcfed99bac
Audited by	© cyberscope

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

Test Deploys

LiberaTokenContract	https://testnet.bscscan.com/address/0x2AE271E84B3D601d463587C44DF4aEa9AC373d00
LiberaPacketContract	https://testnet.bscscan.com/address/0x626c25d699bc7cd2e4481df18f1da740c040b729

Audit Updates

Initial Audit	19 Aug 2024 https://github.com/cyberscope-io/audits/blob/main/libe/v1/audit.pdf
Corrected Phase 2	04 Nov 2024

Source Files

Filename	SHA256
LiberaContract.sol	2cc954b5e79cbb686946f336f76591ef5c6f2d7217c8d07b6f80bebcfed99bac

Overview

LiberaTokenContract (ERC20 Implementation)

The `LiberaTokenContract` is an ERC20 token designed to create and manage "LiberaToken" with the symbol "LIBE." Upon deployment, the contract mints a fixed initial supply of 5 billion tokens, which are assigned to the contract deployer. This contract inherits from OpenZeppelin's ERC20 implementation, providing essential token functionalities such as transferring tokens, checking balances, and managing allowances. Its primary purpose is to facilitate transactions within decentralized applications and ecosystems requiring a standard token structure.

LiberaPacketContract

The `LiberaPacketContract` is a specialized contract for creating, managing, and rewarding data packets. Packets are identified by a unique data hash and managed by an owner. Only the contract owner can create new packets, set rewards, and validate packets, ensuring robust control over the packet lifecycle. Each packet includes metadata such as the owner, name, data type, timestamp, validation status, and reward amount.

The rewardPoolWallet address, set upon deployment, is authorized to distribute rewards to packet owners. Reward distribution is conducted in tokens, and only validated packets are eligible for rewards, with safeguards to prevent rewards for invalid or duplicate packets. Key functionalities include adding packets, validating them, setting rewards, and distributing rewards to packet owners. Events are emitted to log important actions, promoting transparency.

Findings Breakdown

Critical	0
Medium	0
Minor / Informative	9

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	8	1	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	CCR	Contract Centralization Risk	Unresolved
	IDI	Immutable Declaration Improvement	Unresolved
	MU	Modifiers Usage	Unresolved
	MRD	Multiple Reward Distribution	Acknowledged
	TSI	Tokens Sufficiency Insurance	Unresolved
	UVI	Unused Variable Initialization	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L11	Unnecessary Boolean equality	Unresolved
	L16	Validate Variable Setters	Unresolved

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	LiberaContract.sol#L69,111,125,137
Status	Unresolved

Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

Specifically, the owner has the ability to set the characteristics of data packets, determine the rewards associated with them, and validate or invalidate these packets.

Additionally, the `rewardPoolWallet` has the authority to distribute the reward tokens.

```
  function addPacket(
        ...
        packets[_nextPacketID] = PacketStruct(
            _nextPacketID,
            _owner,
            _name,
            _dataHash,
            _dataType,
            block.timestamp,
            false,
            0,
            0
        );

        emit PacketAddedEvent(_owner, _dataHash, _dataType);

        dataHashToPacketId[_dataHash] = _nextPacketID;
        _nextPacketID++;
    }

  function validatePacket(uint256 _id, bool _isValid) external onlyOwner
{
        // This method is used to set the validated flag of a packet by
the owner, which can be either true or false.

        if (packets[_id].id != _id) revert packetDoesNotExistError();

        packets[_id].validated = _isValid;

        emit PacketValidatedEvent(_id, _isValid);

        if (!_isValid) {
            packets[_id].reward = 0;
        }
    }

    function setReward(uint256 _id, uint256 _reward) external onlyOwner
{
        // This method is used to set the reward value of a packet by
the owner.

        if (_reward == 0) revert rewardShouldNotBeZeroError();
        if (packets[_id].id != _id) revert packetDoesNotExistError();
        if (packets[_id].validated == false) revert
packetIsNotValidError();

        packets[_id].reward = _reward;

        emit RewardSetEvent(_id, _reward);
    }
```

```
    function distributeReward(uint256 _id) external onlyRewardPool {
        ...

        bool successRewardTransfer = token.transferFrom(
            rewardPoolWallet,
            packets[_id].owner,
            packets[_id].reward
        );

        ...
    }
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	LiberaContract.sol#L59,61
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
rewardPoolWallet
deployTime
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

MU - Modifiers Usage

Criticality	Minor / Informative
Location	LiberaContract.sol#L114,128,141
Status	Unresolved

Description

The contract is using repetitive statements on some methods to validate some preconditions. In Solidity, the form of preconditions is usually represented by the modifiers. Modifiers allow you to define a piece of code that can be reused across multiple functions within a contract. This can be particularly useful when you have several functions that require the same checks to be performed before executing the logic within the function.

```
if (packets[_id].id != _id) revert packetDoesNotExistError();
...
if (_reward == 0) revert rewardShouldNotBeZeroError();
if (packets[_id].id != _id) revert packetDoesNotExistError();
if (packets[_id].validated == false) revert packetIsNotValidError();
...
if (packets[_id].validated == false) revert packetIsNotValidError();
if (packets[_id].reward == 0) revert rewardShouldNotBeZeroError();
```

Recommendation

The team is advised to use modifiers since it is a useful tool for reducing code duplication and improving the readability of smart contracts. By using modifiers to perform these checks, it reduces the amount of code that is needed to write, which can make the smart contract more efficient and easier to maintain.

MRD - Multiple Reward Distribution

Criticality	Minor / Informative
Location	LiberaContract.sol#L137
Status	Acknowledged

Description

The contract is designed with a `distributeReward` function that allows for the distribution of rewards based on a specific packet ID. However, the function does not include safeguards to prevent multiple executions for the same packet ID. As a result, if the function is invoked multiple times with the same ID, the contract will distribute rewards multiple times, leading to potential overpayment and depletion of the reward pool.

```
    function distributeReward(uint256 _id) external onlyRewardPool {
        //  This method is used for distributing the packet's reward to
the packet's owner.
        // It may be called multiple times since the packets can be sold
multiple times.

        if (packets[_id].validated == false) revert
packetIsNotValidError();
        if (packets[_id].reward == 0) revert
rewardShouldNotBeZeroError();

        bool successRewardTransfer = token.transferFrom(
            rewardPoolWallet,
            packets[_id].owner,
            packets[_id].reward
        );

        if (successRewardTransfer) {
            packets[_id].rewarded += packets[_id].reward;
            emit RewardDistributedEvent(_id, packets[_id].reward);
        }
    }
```

Recommendation

It is recommended to implement checks within the `distributeReward` function to prevent multiple distributions for the same packet ID. This could include adding a flag or condition that ensures the function cannot be executed again for a packet once the rewards have been distributed. This will protect against unintended multiple reward distributions and maintain the integrity of the reward system.

TSI - Tokens Sufficiency Insurance

Criticality	Minor / Informative
Location	LiberaContract.sol#L125,144
Status	Unresolved

Description

The tokens are not held within the contract itself. Instead, the contract is designed to provide the tokens from an external administrator. While external administration can provide flexibility, it introduces a dependency on the administrator's actions, which can lead to various issues and centralization risks.

```
    function setReward(uint256 _id, uint256 _reward) external onlyOwner {
        // This method is used to set the reward value of a packet by
the owner.

        if (_reward == 0) revert rewardShouldNotBeZeroError();
        if (packets[_id].id != _id) revert packetDoesNotExistError();
        if (packets[_id].validated == false) revert
packetIsNotValidError();

        packets[_id].reward = _reward;

        emit RewardSetEvent(_id, _reward);
    }

    function distributeReward(uint256 _id) external onlyRewardPool {
        ...
        bool successRewardTransfer = token.transferFrom(
            rewardPoolWallet,
            packets[_id].owner,
            packets[_id].reward
        );
        ...
        }
```

Recommendation

It is recommended to consider implementing a more decentralized and automated approach for handling the contract tokens. One possible solution is to hold the tokens within the contract itself. If the contract guarantees the process it can enhance its reliability, security, and participant trust, ultimately leading to a more successful and efficient process.

UVI - Unused Variable Initialization

Criticality	Minor / Informative
Location	LiberaContract.sol#L69
Status	Unresolved

Description

The contract contains a `addPacket` function that initializes several variables, including `name` and `dataType`, to set the characteristics of a packet. However, these variables are not utilized elsewhere in the contract, rendering them redundant. This unnecessary initialization increases gas consumption without providing any functional benefit, leading to inefficiencies in the contract.

```
   function addPacket(
        address _owner,
        string calldata _name,
        string calldata _dataHash,
        string calldata _dataType
    ) external onlyOwner {
        // This method is used for packet creation by the owner.
        // typically called from the main application with the owner's
authorisation.

        if (dataHashToPacketId[_dataHash] != 0) revert
packetDoesExistError();

        packets[_nextPacketID] = PacketStruct(
            _nextPacketID,
            _owner,
            _name,
            _dataHash,
            _dataType,
            block.timestamp,
            false,
            0,
            0
        );
```

Recommendation

It is recommended to reconsider the variables required for the initialization of packets. If these variables, are not intended to be used within the contract or in future implementations, they could be removed to reduce gas consumption and streamline the contract's code.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	LiberaContract.sol#L70,71,72,73,98,111,125,137
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
address _owner
string calldata _name
string calldata _dataHash
string calldata _dataType
uint256 _id
bool _isValid
uint256 _reward
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.
Find more information on the Solidity documentation
https://docs.soliditylang.org/en/stable/style-guide.html#naming-conventions.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	LiberaContract.sol#L130,141
Status	Unresolved

Description

Boolean equality is unnecessary when comparing two boolean values. This is because a boolean value is either true or false, and there is no need to compare two values that are already known to be either true or false.

it's important to be aware of the types of variables and expressions that are being used in the contract's code, as this can affect the contract's behavior and performance. The comparison to boolean constants is redundant. Boolean constants can be used directly and do not need to be compared to true or false.

```
packets[_id].validated == false
```

Recommendation

Using the boolean value itself is clearer and more concise, and it is generally considered good practice to avoid unnecessary boolean equalities in Solidity code.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	LiberaContract.sol#L59
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
rewardPoolWallet = _rewardPoolWallet
```

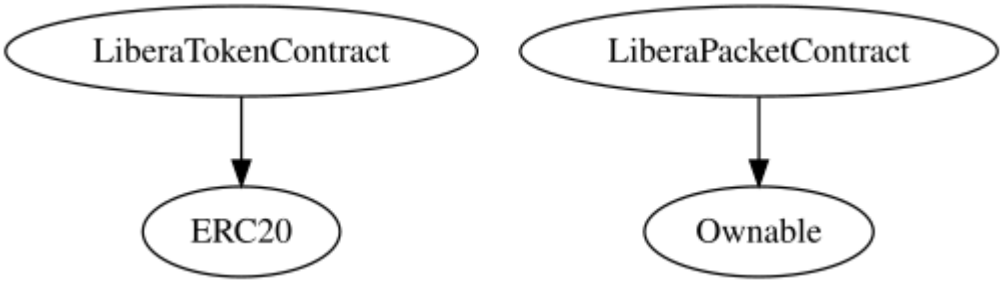
Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

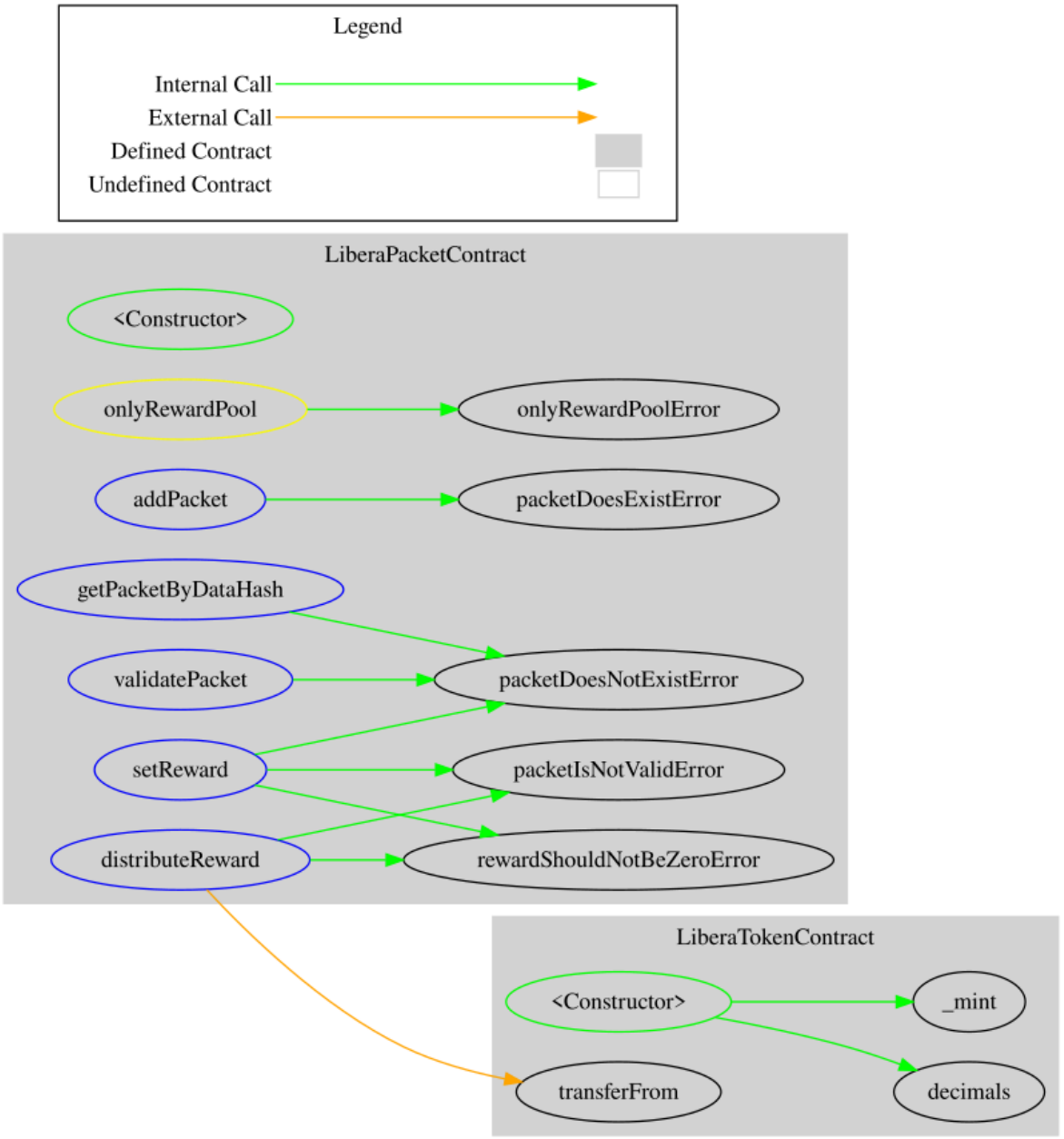
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
LiberaTokenContract	Implementation	ERC20		
		Public	✓	ERC20
LiberaPacketContract	Implementation	Ownable		
		Public	✓	Ownable
	addPacket	External	✓	onlyOwner
	getPacketByDataHash	External		-
	validatePacket	External	✓	onlyOwner
	setReward	External	✓	onlyOwner
	distributeReward	External	✓	onlyRewardPool

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
LiberaPacketContract
<Constructor>
onlyRewardPool
onlyRewardPoolError
addPacket
packetDoesExistError
getPacketByDataHash
validatePacket
packetDoesNotExistError
setReward
packetIsNotValidError
distributeReward
rewardShouldNotBeZeroError
LiberaTokenContract
<Constructor>
_mint
transferFrom
decimals

Summary

The LiberaPacketContract integrates tokenized rewards and data management functionalities. This audit examines the security, business logic, and scalability of the contract, focusing on ensuring efficient and secure reward distribution. LiberaPacket is a promising project with a structured approach to packet creation, validation, and reward allocation, backed by a reliable ERC20-based token, LiberaToken. The analysis found no compiler errors or critical vulnerabilities. The contract Owner has privileged access to specific administrative functions, but these are designed with safeguards to prevent misuse or disruption of user interactions.

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