Cyberscope

Audit Report

FORTUNA

May 2024

Network ETH

Address 0x329c1a5d068726b53fb874cbd1e476e514a664ab

Audited by © cyberscope

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	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
	CCR	Contract Centralization Risk	Unresolved
	IDI	Immutable Declaration Improvement	Unresolved
	MEE	Missing Events Emission	Unresolved
	RV	Randomization Vulnerability	Unresolved
	RRS	Redundant Require Statement	Unresolved
	RSML	Redundant SafeMath Library	Unresolved
	RSW	Redundant Storage Writes	Unresolved
	UAR	Unexcluded Address Restrictions	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L07	Missing Events Arithmetic	Unresolved
	L09	Dead Code Elimination	Unresolved
	L11	Unnecessary Boolean equality	Unresolved
	L16	Validate Variable Setters	Unresolved
	L18	Multiple Pragma Directives	Unresolved

Table of Contents

Review

Contract Name	FORTUNA
Compiler Version	v0.8.25+commit.b61c2a91
Optimization	200 runs
Explorer	https://etherscan.io/address/0x329c1a5d068726b53fb874cbd1e476e514a664ab
Address	0x329c1a5d068726b53fb874cbd1e476e514a664ab
Network	ETH
Symbol	FRTN8
Decimals	18
Total Supply	100,000,000
Badge Eligibility	Must Fix Criticals

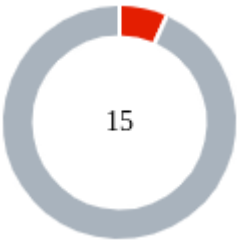
Audit Updates

Initial Audit	02 May 2024

Source Files

Filename	SHA256
FORTUNA.sol	b763eefe64ebacc12e55f3ae111ecc884cb359f9b29135e15641c7b49e751175

Findings Breakdown



Critical	1
Medium	0
Minor / Informative	14

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	0	0	0	0
Minor / Informative	14	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	FORTUNA.sol#L1242,1303
Status	Unresolved

Description

The contract owner has the authority to stop the sales for all users excluding the owner. The owner may take advantage of it by setting the `vault` to the pair address. As a result, the contract may operate as a honeypot.

Furthermore, if the holders array is empty or the vault address is zero (L16), the transactions will revert.

```
require(!(recipient == vault || recipient == ownerCntr), "Cannot send to vault
or owner");
...
function setVault(address _vault) external onlyOwner {
    vault = _vault;
}
...
require(holders.length > 0, "No holders with enough tokens");
```

Recommendation

The contract could embody a check for not allowing setting the `_maxTxAmount` less than a reasonable amount. A suggested implementation could check that the minimum amount should be more than a fixed percentage of the total supply. The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.

- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	FORTUNA.sol#L1303,1307,1312,1316,1320
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.

```
function setVault(address _vault) external onlyOwner
function setTaxPercentage(uint256 _taxPercentage) external onlyOwner
function excludeFromTax(address account) external onlyOwner
function includeInTax(address account) external onlyOwner
function setlotteryRestriction(address account, bool restricted)
external onlyOwner
```

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	FORTUNA.sol#L1223
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.

```
ownerCntr
```

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.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	FORTUNA.sol#L1303,1312,1316,1320
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
function setVault(address _vault) external onlyOwner
function excludeFromTax(address account) external onlyOwner
function includeInTax(address account) external onlyOwner
function setlotteryRestriction(address account, bool restricted)
external onlyOwner
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

RV - Randomization Vulnerability

Criticality	Minor / Informative
Location	FORTUNA.sol#L1286
Status	Unresolved

Description

The contract is using an on-chain technique in order to determine random numbers. The blockchain runtime environment is fully deterministic, as a result, the pseudo-random numbers could be predicted.

```
uint256 index = uint256(keccak256(abi.encodePacked(block.timestamp,
block.prevrandao))) % holders.length;
```

Recommendation

The contract could use an advanced randomization technique that guarantees an acceptable randomization factor. For instance, the Chainlink VRF (Verifiable Random Function). https://docs.chain.link/docs/chainlink-vrf

RRS - Redundant Require Statement

Criticality	Minor / Informative
Location	FORTUNA.sol#L1288
Status	Unresolved

Description

In the `triggerLottery` function, there is a `require` statement checking if the `vaultBalance` is greater than or equal to `VAULT_THRESHOLD`. This check ensures that the vault balance is sufficient to proceed with the lottery. However, this same check is already performed within the `_transfer` function before calling `triggerLottery`. The `_transfer` function verifies if the vault balance exceeds the threshold before invoking `triggerLottery`.

```
// Check if vault balance exceeds threshold for lottery
if (balanceOf(vault) >= VAULT_THRESHOLD) {
    triggerLottery();
}
...
require(vaultBalance >= VAULT_THRESHOLD, "Vault balance is not enough
for lottery");
```

Recommendation

The team is advised to remove the redundant `require` statement in the `triggerLottery` function, to optimize gas usage and improve contract efficiency. By relying on the pre-existing check in the `_transfer` function, the contract's execution cost can be reduced.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	FORTUNA.sol
Status	Unresolved

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert to underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases gas consumption unnecessarily in cases where the explanatory error message is not used.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library in cases where the revert error message is not used. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	FORTUNA.sol#L1320
Status	Unresolved

Description

The contract modifies the state of the following variables without checking if their current value is the same as the one given as an argument. As a result, the contract performs redundant storage writes, when the provided parameter matches the current state of the variables, leading to unnecessary gas consumption and inefficiencies in contract execution.

```
    function setlotteryRestriction(address account, bool restricted)
external onlyOwner {
        lotteryRestriction[account] = restricted;
    }
```

Recommendation

The team is advised to implement additional checks within to prevent redundant storage writes when the provided argument matches the current state of the variables. By incorporating statements to compare the new values with the existing values before proceeding with any state modification, the contract can avoid unnecessary storage operations, thereby optimizing gas usage.

UAR - Unexcluded Address Restrictions

Criticality	Minor / Informative
Location	FORTUNA.sol#L1244
Status	Unresolved

Description

The contract incorporates operational restrictions on transactions, which can hinder seamless interaction with decentralized applications (dApps) such as launchpads, presales, lockers, or staking platforms. In scenarios where an external contract, such as a launchpad factory, needs to integrate with the contract, it should be exempt from the limitations to ensure uninterrupted service and functionality. Failure to provide such exemptions can block the successful process and operation of services reliant on this contract.

```
if (!isExcludedFromTax[sender]) {
    taxAmount = amount.mul(taxPercentage).div(100);
    super._transfer(sender, vault, taxAmount);
}
```

Recommendation

It is advisable to modify the contract by incorporating functionality that enables the exclusion of designated addresses from transactional restrictions. This enhancement will allow specific addresses, such as those associated with decentralized applications (dApps) and service platforms, to operate without being hindered by the standard constraints imposed on other users. Implementing this feature will ensure smoother integration and functionality with external systems, thereby expanding the contract's versatility and effectiveness in diverse operational environments.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	FORTUNA.sol#L1303,1307
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.

```
_vault
_taxPercentage
```

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/v0.8.17/style-guide.html#naming-convention.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	FORTUNA.sol#L1307
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
    function setTaxPercentage(uint256 _taxPercentage) external onlyOwner
{
        require(_taxPercentage <= 10, "Tax percentage exceeds 10%");
        taxPercentage = _taxPercentage;
    }
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	FORTUNA.sol#L43,72,85,97,106,116,137,148,456,651,1109
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function sendValue(address payable recipient, uint256 amount) internal {
        if (address(this).balance < amount) {
            revert AddressInsufficientBalance(address(this));
        }

        (bool success, ) = recipient.call{value: amount}("");
        if (!success) {
            revert FailedInnerCall();
        }
    }

function functionCall(address target, bytes memory data) internal
returns (bytes memory) {
        return functionCallWithValue(target, data, 0);
    }

...
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	FORTUNA.sol#L1292
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.

```
if(lotteryRestriction[winner]==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	FORTUNA.sol#L1222,1304
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.

```
_vaultAddress;
...
_vault;
```

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.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	FORTUNA.sol#L6,168,386,465,630,661,763,845,873,1189
Status	Unresolved

Description

If the contract includes multiple conflicting pragma directives, it may produce unexpected errors. To avoid this, it's important to include the correct pragma directive at the top of the contract and to ensure that it is the only pragma directive included in the contract.

```
pragma solidity ^0.8.0;
pragma solidity ^0.8.20;
pragma solidity ^0.8.25;
```

Recommendation

It is important to include only one pragma directive at the top of the contract and to ensure that it accurately reflects the version of Solidity that the contract is written in.

By including all required compiler options and flags in a single pragma directive, the potential conflicts could be avoided and ensure that the contract can be compiled correctly.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Address	Library			
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	verifyCallResultFromTarget	Internal		
	verifyCallResult	Internal		
	_revert	Private		
SafeMath	Library			
	tryAdd	Internal		
	trySub	Internal		
	tryMul	Internal		
	tryDiv	Internal		
	tryMod	Internal		
	add	Internal		
	sub	Internal		
	mul	Internal		

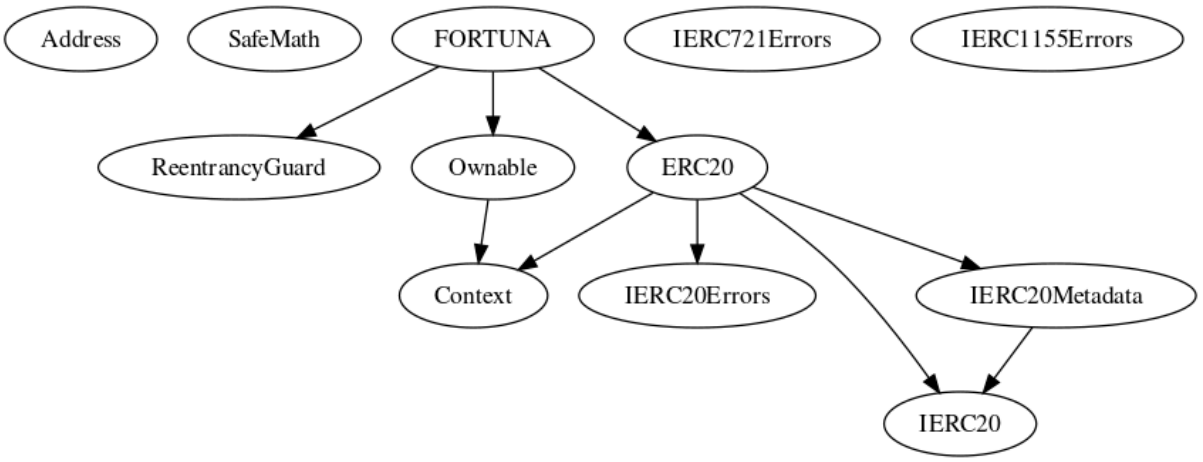
	div	Internal		
	mod	Internal		
	sub	Internal		
	div	Internal		
	mod	Internal		
ReentrancyGuard	Implementation			
		Public	✓	-
	_nonReentrantBefore	Private	✓	
	_nonReentrantAfter	Private	✓	
	_reentrancyGuardEntered	Internal		
IERC20Errors	Interface			
IERC721Errors	Interface			
IERC1155Errors	Interface			
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
	_contextSuffixLength	Internal		

Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
ERC20	Implementation	Context, IERC20, IERC20Metadata,		

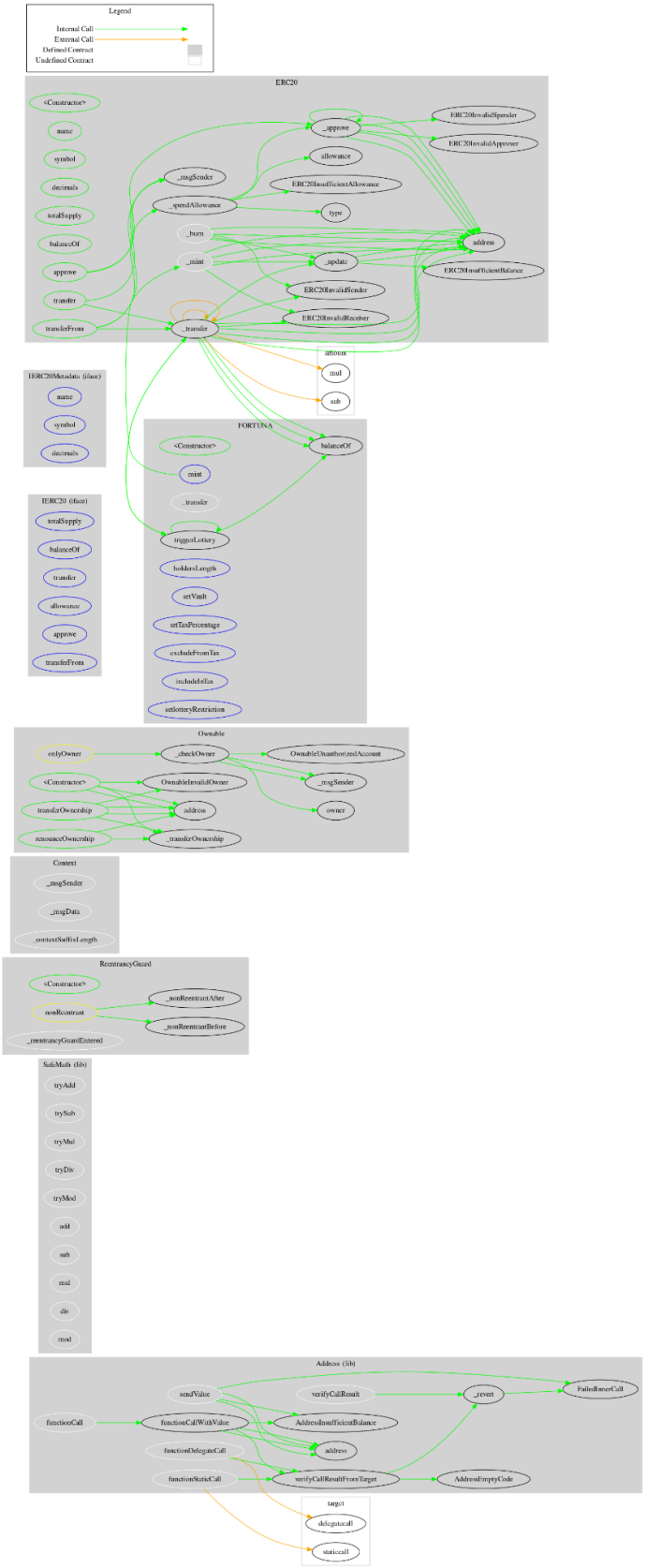
		IERC20Errors		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	_transfer	Internal	✓	
	_update	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_approve	Internal	✓	
	_spendAllowance	Internal	✓	
FORTUNA	Implementation	ERC20, Ownable, ReentrancyGuard		
		Public	✓	ERC20 Ownable
	mint	External	✓	onlyOwner

	_transfer	Internal	✓	
	triggerLottery	Private	✓	nonReentrant
	holdersLength	External	✓	onlyOwner
	setVault	External	✓	onlyOwner
	setTaxPercentage	External	✓	onlyOwner
	excludeFromTax	External	✓	onlyOwner
	includeInTax	External	✓	onlyOwner
	setlotteryRestriction	External	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
approve
transfer
transferFrom
_approve
ERC20InvalidSpender
ERC20InvalidApprover
allowance
_msgSender
ERC20InsufficientAllowance
_spendAllowance
type
_burn
address
_mint
_update
ERC20InsufficientBalance
ERC20InvalidSender
ERC20InvalidReceiver
_transfer
amount
mul
sub
IERC20Metadata (iface)
name
symbol
decimals
FORTUNA
<Constructor>
balanceOf
mint
_transfer
triggerLottery
holdersLength
setVault
setTaxPercentage
excludeFromTax
includeInTax
setlotteryRestriction
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Ownable
onlyOwner
_checkOwner
OwnableUnauthorizedAccount
<Constructor>
OwnableInvalidOwner
_msgSender
transferOwnership
address
owner
renounceOwnership
_transferOwnership
Context
_msgSender
_msgData
_contextSuffixLength
ReentrancyGuard
<Constructor>
_nonReentrantAfter
nonReentrant
_nonReentrantBefore
_reentrancyGuardEntered
SafeMath (lib)
tryAdd
trySub
tryMul
tryDiv
tryMod
add
sub
mul
div
mod
Address (lib)
sendValue
verifyCallResult
_revert
FailedInnerCall
functionCall
functionCallWithValue
AddressInsufficientBalance
address
functionDelegateCall
functionStaticCall
verifyCallResultFromTarget
AddressEmptyCode
target
delegatecall
staticcall

Summary

FORTUNA is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues. The contract Owner can access some admin functions that can not be used in a malicious way to disturb the users' transactions. There is also a limit of max 10% fees. This audit investigates security issues, business logic concerns and potential improvements.

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

https://www.cyberscope.io