Cyberscope

Audit Report

The Aurora Foundation

May 2024

SHA256 a8ac01413c6d6b7df7680caffd90c9ee66fba5a300ead9e2d600ad6485f016dd

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
●	FAAL	Flawed Admin Authentication Logic	Unresolved
●	CO	Code Optimization	Unresolved
●	CR	Code Repetition	Unresolved
●	MNC	Misleading Naming Convention	Unresolved
●	MSLI	Missing SPDX License Identifier	Unresolved
●	PAMAR	Pair Address Max Amount Restriction	Unresolved
●	UAR	Unexcluded Address Restrictions	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L18	Multiple Pragma Directives	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Analysis **1**
Diagnostics **2**
Table of Contents **3**
Review **5**
Audit Updates 5
Source Files 5
Findings Breakdown **6**
ST - Stops Transactions 7
Description 7
Recommendation 8
FAAL - Flawed Admin Authentication Logic 9
Description 9
Recommendation 9
CO - Code Optimization 10
Description 10
Recommendation 11
CR - Code Repetition 12
Description 12
Recommendation 13
MNC - Misleading Naming Convention 14
Description 14
Recommendation 15
MSLI - Missing SPDX License Identifier 16
Description 16
Recommendation 16
PAMAR - Pair Address Max Amount Restriction 17
Description 17
Recommendation 17
UAR - Unexcluded Address Restrictions 18
Description 18
Recommendation 18
L04 - Conformance to Solidity Naming Conventions 19
Description 19
Recommendation 19
L09 - Dead Code Elimination 20
Description 20
Recommendation 20
L18 - Multiple Pragma Directives 21
Description 21

Recommendation 21
L19 - Stable Compiler Version 22
Description 22
Recommendation 22
Functions Analysis 23
Inheritance Graph 27
Flow Graph 28
Summary 29
Disclaimer 30
About Cyberscope 31

Review

Contract Name	AuroraFoundation
Testing Deploy	https://testnet.bscscan.com/address/0xec46e6ad8ec5a0eb3cfa0ce44f571fd87ca7e0dc
Symbol	AUF
Decimals	18
Total Supply	100,000,000
Badge Eligibility	Must Fix Criticals

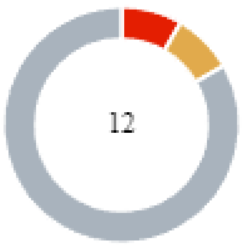
Audit Updates

Initial Audit	04 May 2024

Source Files

Filename	SHA256
contracts/Auf_audit.sol	a8ac01413c6d6b7df7680caffd90c9ee66fba5a300ead9e2d600ad6485f016dd

Findings Breakdown



●	Critical	1
●	Medium	1
●	Minor / Informative	10

Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	1	0	0	0
●	Medium	1	0	0	0
●	Minor / Informative	10	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	contracts/Auf_audit.sol#L747,809,814,831
Status	Unresolved

Description

The contract owner has the authority to stop the sales for all users excluding the addresses that belong in the `excludedAddresses` mapping, which is a mapping managed by the contract owner. The owner may take advantage of it by calling the `freezeAll` function. As a result, the contract may operate as a honeypot. Additionally, the contract owner has the authority to stop the sale by setting the `maxPerWalletPercentage` and the `maxPerTxPercentage` to zero. Lastly, the contract owner has the authority to stop the sales, as described in the PAMAR finding.

```
function freezeAll() public onlyAdmin {
    paused = true;
}

modifier notFrozen(address _from, address _to) {
    require(!paused || excludedAddresses[_from] ||
excludedAddresses[_to], "Transactions are paused");
    _;
}

function setMaxPerWalletPercentage(uint256 percentage) public
onlyAdmin {
    require(percentage <= MAX_PERCENTAGE_SUPPLY, "Percentage
exceeds maximum");
    maxPerWalletPercentage = percentage;
}

function setMaxPerTxPercentage(uint256 percentage) public
onlyAdmin {
    require(percentage <= MAX_PERCENTAGE_SUPPLY, "Percentage
exceeds maximum");
    maxPerTxPercentage = percentage;
}
```

Recommendation

The contract could embody a check for not allowing setting the `maxPerTxPercentage` and the `maxPerWalletPercentage` 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. In order for this solution to resolve the issue, the FAAL finding should be address as well.

FAAL - Flawed Admin Authentication Logic

Criticality	Medium
Location	contracts/Auf_audit.sol#L742
Status	Unresolved

Description

The `onlyAdmin` modifier allows function execution if the caller is the owner of the contract or if the owner address has been set to zero. This logic inherently means that if the contract's ownership is renounced, all functions protected by this modifier become public, allowing anyone to call these functions. Functions that have this modifier, which are critical to the security and operational integrity of the contract, would thus be exposed to unrestricted access, potentially leading to unintended behaviors within the contract.

```
modifier onlyAdmin() {
    require(_msgSender() == owner() || owner() == address(0),
"Caller is not the admin");
    _;
}
```

Recommendation

it is recommended to revise the modifier’s logic to ensure continued control over the functions it protects. Instead of allowing unrestricted access when the owner address is set to zero, the contract should enforce that only an explicitly defined administrator can call these sensitive functions.

CO - Code Optimization

Criticality	Minor / Informative
Location	contracts/Auf_audit.sol#L776,785,794
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations. Specifically, the `safeTransfer` and `safeTransferFrom` don't use the `_beforeTokenTransfer` internal function and have the checks implemented again.

```
function safeTransfer(address recipient, uint256 amount) public
virtual notFrozen(_msgSender(), recipient) {
    require(block.number != lastTxBlock[_msgSender()], "Only
one transaction per block allowed");
    require(amount <= maxPerTx(), "Transfer amount exceeds the
maximum per transaction");
    require(balanceOf(recipient) + amount <= maxPerWallet(),
"Recipient balance would exceed the maximum per wallet");

    lastTxBlock[_msgSender()] = block.number;
    super._transfer(_msgSender(), recipient, amount);
}

function safeTransferFrom(address sender, address recipient,
uint256 amount) public virtual notFrozen(sender, recipient) {
    require(block.number != lastTxBlock[sender], "Only one
transaction per block allowed");
    require(amount <= maxPerTx(), "Transfer amount exceeds the
maximum per transaction");
    require(balanceOf(recipient) + amount <= maxPerWallet(),
"Recipient balance would exceed the maximum per wallet");

    lastTxBlock[sender] = block.number;
    super._transfer(sender, recipient, amount);
}

function _beforeTokenTransfer(address from, address to, uint256
amount) internal virtual {
    require(block.number != lastTxBlock[from], "Only one
transaction per block allowed");
    require(amount <= maxPerTx(), "Transfer amount exceeds the
maximum per transaction");
    require(balanceOf(to) + amount <= maxPerWallet(),
"Recipient balance would exceed the maximum per wallet");
    lastTxBlock[from] = block.number;
}
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

CR - Code Repetition

Criticality	Minor / Informative
Location	contracts/Auf_audit.sol#819,823,843,847 contracts/Auf_audit.sol#L766,771,776,785,794
Status	Unresolved

Description

The contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible. Specifically, the following functions serve the same purpose.

```
function excludeAddress(address _address) public onlyAdmin {
    excludedAddresses[_address] = true;
}

function includeAddress(address _address) public onlyAdmin {
    excludedAddresses[_address] = false;
}

function freezeAddress(address _address) public onlyAdmin {
    excludedAddresses[_address] = true;
}

function unfreezeAddress(address _address) public onlyAdmin {
    excludedAddresses[_address] = false;
}
```

Furthermore, the "safe" transfers are essentially replicating the functionality of `transfer` and `transferFrom` functions, especially with the way `_beforeTokenTransfer` is structured.

```
function transfer(address recipient, uint256 amount) public
virtual override notFrozen(_msgSender(), recipient) returns
(bool) {
    _beforeTokenTransfer(_msgSender(), recipient, amount);
    return super.transfer(recipient, amount);
}

function safeTransfer(address recipient, uint256 amount) public
virtual notFrozen(_msgSender(), recipient) {
    require(block.number != lastTxBlock[_msgSender()], "Only
one transaction per block allowed");
    require(amount <= maxPerTx(), "Transfer amount exceeds the
maximum per transaction");
    require(balanceOf(recipient) + amount <= maxPerWallet(),
"Recipient balance would exceed the maximum per wallet");

    lastTxBlock[_msgSender()] = block.number;
    super._transfer(_msgSender(), recipient, amount);
}

function _beforeTokenTransfer(address from, address to, uint256
amount) internal virtual {
    require(block.number != lastTxBlock[from], "Only one
transaction per block allowed");
    require(amount <= maxPerTx(), "Transfer amount exceeds the
maximum per transaction");
    require(balanceOf(to) + amount <= maxPerWallet(),
"Recipient balance would exceed the maximum per wallet");
    lastTxBlock[from] = block.number;
}
```

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help reduce the complexity and size of the contract. For instance, the contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places.

MNC - Misleading Naming Convention

Criticality	Minor / Informative
Location	contracts/Auf_audit.sol#L747,843,847,851
Status	Unresolved

Description

Several functions and a modifier use misleading naming conventions that could cause confusion regarding their actual functionality. The functions `freezeAddress` and `unfreezeAddress` do not freeze or unfreeze an address in the traditional sense. Instead, they set the status of an address in the `excludedAddresses` mapping, which effectively whitelists addresses from a global pause imposed by the freezeAll function. Contrary to what their names suggest, setting an address with `freezeAddress` allows it to continue transactions even when the contract is globally paused, and `unfreezeAddress` removes this capability. Similarly, the function `isAddressFrozen` misleadingly suggests it would return true if an address is prohibited from making transactions (frozen). In reality, it returns true if the address is exempt from such restrictions (whitelisted). The modifier `notFrozen` also utilizes this mapping to allow transactions involving at least one whitelisted address, which could further confuse the interpretation of what being "frozen" entails in this context.

```
modifier notFrozen(address _from, address _to) {
     require(!paused || excludedAddresses[_from] ||
excludedAddresses[_to], "Transactions are paused");
     _;
}

function freezeAddress(address _address) public onlyAdmin {
     excludedAddresses[_address] = true;
}

function unfreezeAddress(address _address) public onlyAdmin {
     excludedAddresses[_address] = false;
}

function isAddressFrozen(address _address) public view returns
(bool) {
     return excludedAddresses[_address];
}
```

Recommendation

To prevent misunderstanding and ensure clarity in contract functionality, it is recommended to rename these functions and the modifier to accurately reflect their operations. Adopting these changes will enhance the readability and comprehensibility of the contract, reducing potential errors in its use.

MSLI - Missing SPDX License Identifier

Criticality	Minor / Informative
Location	contracts/Auf_audit.sol#L1
Status	Unresolved

Description

The smart contract lacks an SPDX (Software Package Data Exchange) license identifier, which is a standard practice for specifying the license under which the software is made available. This identifier is crucial for defining the legal conditions under which the software can be used, modified, and shared. The absence of an SPDX identifier can lead to ambiguity regarding the copyright and licensing status of the contract. It also exposes the contract to legal and compliance risks, especially in environments where open-source software compliance is strictly monitored.

Recommendation

It is recommended to add an SPDX license identifier at the beginning of the source file. The identifier should reflect the intended usage rights of the code. If the code is intended for open-source use, an appropriate open-source license should be selected and declared. Conversely, if the code is not intended to be freely available for reuse and modification, it should be marked as "UNLICENSED" to explicitly denote that it is not open-source. Including an SPDX identifier will clarify the licensing terms, enhance the project's legal clarity, and ensure alignment with best practices for software distribution and compliance.

PAMAR - Pair Address Max Amount Restriction

Criticality	Minor / Informative
Location	contracts/Auf_audit.sol#L779,788,797
Status	Unresolved

Description

The contract is configured to enforce a maximum token accumulation limit through checks. This mechanism aims to prevent excessive token concentration by reverting transactions that overcome the specified cap. However, this functionality encounters issues when transactions default to the pair address during sales. If the pair address is not listed in the exceptions, then the sale transactions are inadvertently stopped, effectively disrupting operations and making the contract susceptible to unintended behaviors akin to a honeypot.

```
require(balanceOf(recipient) + amount <= maxPerWallet(),
"Recipient balance would exceed the maximum per wallet");
```

Recommendation

It is advised to modify the contract to ensure uninterrupted operations by either permitting the pair address to exceed the established token accumulation limit or by safeguarding its status in the exception list. By recognizing and allowing these essential addresses the flexibility to hold more tokens than typical limits, the contract can maintain seamless transaction flows and uphold the liquidity and stability of the ecosystem. This modification is vital for avoiding disruptions that could impact the functionality and security of the contract.

UAR - Unexcluded Address Restrictions

Criticality	Minor / Informative
Location	contracts/Auf_audit.sol#L778,779,787,788,796,797
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.

```
require(amount <= maxPerTx(), "Transfer amount exceeds the
maximum per transaction");
require(balanceOf(recipient) + amount <= maxPerWallet(),
"Recipient balance would exceed the maximum per wallet");
```

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	contracts/Auf_audit.sol#L819,823,843,847,851
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.

```
function excludeAddress(address _address) public onlyAdmin {

...
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	contracts/Auf_audit.sol#L188,646
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 _contextSuffixLength() internal view virtual returns
(uint256) {
    return 0;
}

function _burn(address account, uint256 value) internal {
    if (account == address(0)) {
        revert ERC20InvalidSender(address(0));
    }
    _update(account, address(0), value);
}
```

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.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	contracts/Auf_audit.sol#L2,167,198,300,382,410,726
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 ^0.8.20;
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/Auf_audit.sol#L2,167,198,300,382,410,726
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 ^0.8.20;
```

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

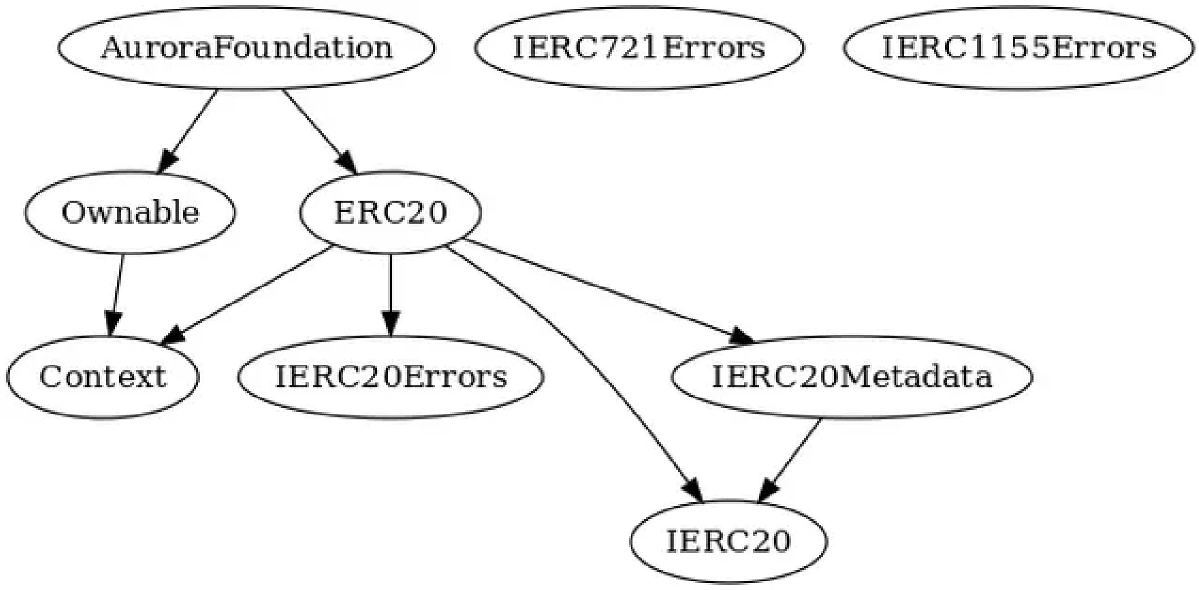
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC20Errors	Interface			
IERC721Errors	Interface			
IERC1155Errors	Interface			
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
	_contextSuffixLength	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	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	✓	-

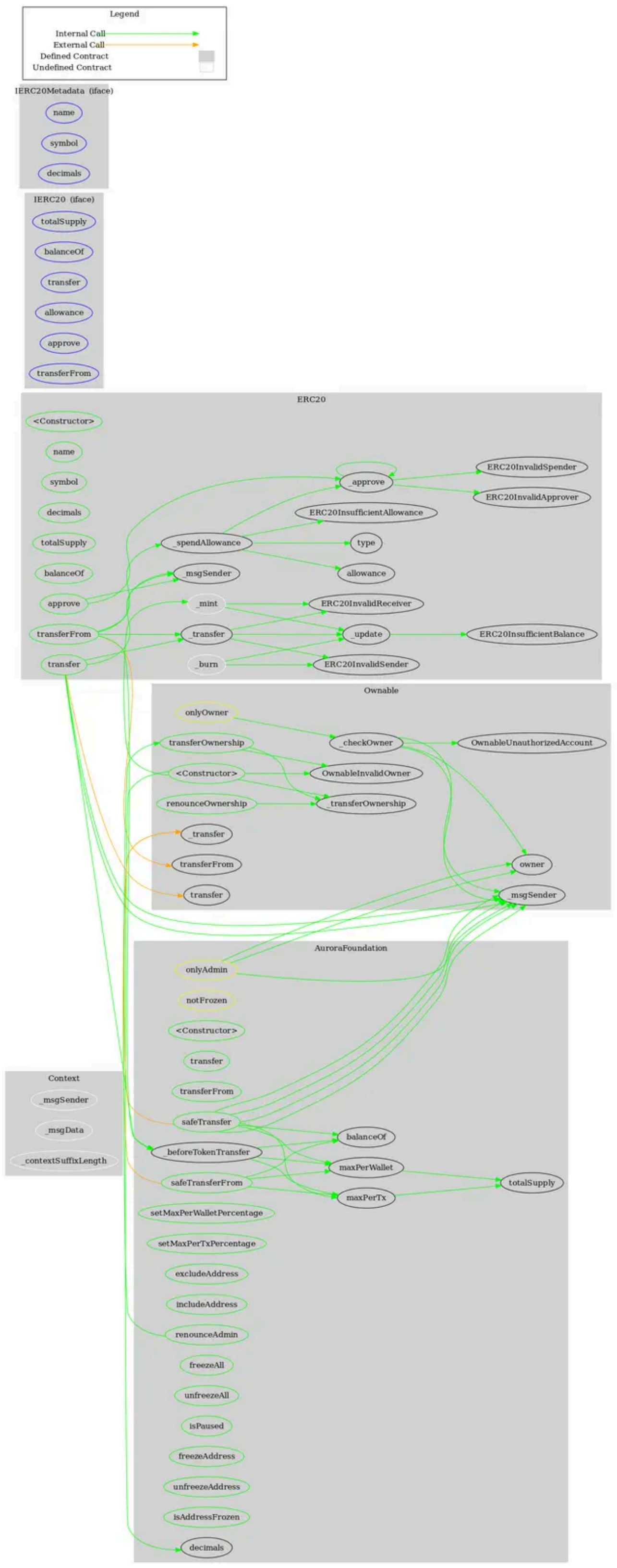
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	_transfer	Internal	✓	
	_update	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_approve	Internal	✓	
	_spendAllowance	Internal	✓	
AuroraFoundation	Implementation	ERC20, Ownable		
		Public	✓	ERC20 Ownable
	transfer	Public	✓	notFrozen
	transferFrom	Public	✓	notFrozen
	safeTransfer	Public	✓	notFrozen
	safeTransferFrom	Public	✓	notFrozen
	_beforeTokenTransfer	Internal	✓	
	maxPerWallet	Public		-
	maxPerTx	Public		-
	setMaxPerWalletPercentage	Public	✓	onlyAdmin
	setMaxPerTxPercentage	Public	✓	onlyAdmin
	excludeAddress	Public	✓	onlyAdmin

	includeAddress	Public	✓	onlyAdmin
	renounceAdmin	Public	✓	onlyAdmin
	freezeAll	Public	✓	onlyAdmin
	unfreezeAll	Public	✓	onlyAdmin
	isPaused	Public		-
	freezeAddress	Public	✓	onlyAdmin
	unfreezeAddress	Public	✓	onlyAdmin
	isAddressFrozen	Public		-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IERC20Metadata (iface)
name
symbol
decimals
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
approve
transferFrom
transfer
_approve
_spendAllowance
_msgSender
_mint
_transfer
_burn
_update
type
allowance
ERC20InvalidSpender
ERC20InvalidApprover
ERC20InsufficientAllowance
ERC20InvalidReceiver
ERC20InvalidSender
ERC20InsufficientBalance
Ownable
onlyOwner
transferOwnership
<Constructor>
renounceOwnership
transfer
transferFrom
transfer
_checkOwner
OwnableInvalidOwner
_transferOwnership
OwnableUnauthorizedAccount
owner
_msgSender
AuroraFoundation
onlyAdmin
notFrozen
<Constructor>
transfer
transferFrom
safeTransfer
beforeTokenTransfer
safeTransferFrom
setMaxPerWalletPercentage
setMaxPerTxPercentage
excludeAddress
includeAddress
renounceAdmin
freezeAll
unfreezeAll
isPaused
freezeAddress
unfreezeAddress
isAddressFrozen
decimals
balanceOf
maxPerWallet
maxPerTx
totalSupply
Context
_msgSender
_msgData
_contextSuffixLength

Summary

The Aurora Foundation contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. There are some functions that can be abused by the owner like stop transactions. A multi-wallet signing pattern will provide security against potential hacks.

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