Cyberscope

Audit Report

The Aurora Foundation

May 2024

SHA256 2fa85157e9bdb62bb2aa4c712ef90d3575c0aa824efe3152d1f046f13ccbab0a

Audited by © cyberscope

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Acknowledged
●	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
●	CR	Code Repetition	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	MNC	Misleading Naming Convention	Unresolved
●	PAMAR	Pair Address Max Amount Restriction	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Review

Contract Name	AuroraFoundation
Testing Deploy	https://testnet.bscscan.com/address/0xc9917c8c9dd754e828f6672dbb18ce869a898058
Symbol	AUF
Decimals	18
Total Supply	100,000,000
Badge Eligibility	Must Fix Criticals

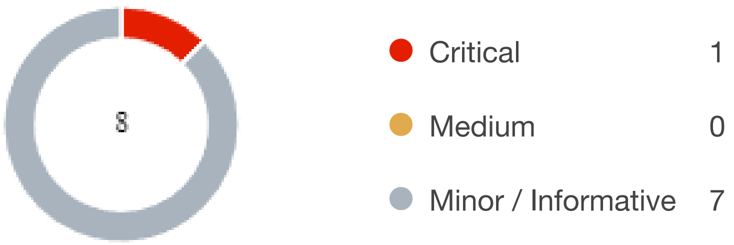
Audit Updates

Initial Audit	04 May 2024 https://github.com/cyberscope-io/audits/blob/main/1-auf/v1/audit.pdf
Corrected Phase 2	17 May 2024

Source Files

Filename	SHA256
contracts/ERC20.sol	9c40cb99df0d04cd2a45397b2f6a154bf9faa7a25b06203c815f8748e879a012
contracts/AUF.sol	2fa85157e9bdb62bb2aa4c712ef90d3575c0aa824efe3152d1f046f13ccbab0a

Findings Breakdown



	Severity	Unresolved	Acknowledged	Resolved	Other
●	Critical	0	1	0	0
●	Medium	0	0	0	0
●	Minor / Informative	7	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	contracts/AUF.sol#L22,82
Status	Acknowledged

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 for the first 30 days after the contract has been deployed. The owner may take advantage of it by calling the `freezeAll` function. As a result, the contract may operate as a honeypot. Lastly, the contract owner has the authority to stop the sales, as described in the PAMAR finding.

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

modifier notFrozen(address _from, address _to) {
    require(!paused || excludedAddresses[_from] ||
excludedAddresses[_to] || block.timestamp > deploymentTimestamp
+ UNFREEZE_DELAY, "Transactions are paused");
    _;
}
```

Recommendation

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.

Team Update

The team states: *This is meant to work that way, we want the restrictions to be removed after a certain delay independently from the owner's will. This means, that after deployment, even if the admin decides to not unfreeze the transfers, which will make the contract behave like a honeypot, the freeze will be removed automatically after a certain delay.*

CR - Code Repetition

Criticality	Minor / Informative
Location	contracts/AUF.sol#L70,74,90,94
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.

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

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

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

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

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.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	contracts/AUF.sol#L30
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.

```
deploymentTimestamp
```

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.

MNC - Misleading Naming Convention

Criticality	Minor / Informative
Location	contracts/AUF.sol#L22,90,94,98
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] || block.timestamp > deploymentTimestamp
+ UNFREEZE_DELAY, "Transactions are paused");
    _;
}

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

function unfreezeAddress(address _address) public onlyOwner {
    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.

PAMAR - Pair Address Max Amount Restriction

Criticality	Minor / Informative
Location	contracts/AUF.sol#L43,74
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. Furthermore, even if it is excluded from the restrictions, the contract owner can enforce the restrictions again by calling the `includeAddress` function.

```
function _beforeTokenTransfer(address from, address to, uint256
amount) internal virtual {
    if(!(excludedAddresses[from] || excludedAddresses[to])){
        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;
    }
}

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

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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/AUF.sol#L70,74,90,94,98
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 _address
```

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	contracts/AUF.sol#L62,67
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.

```
maxPerWalletPercentage = percentage
maxPerTxPercentage = percentage
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/AUF.sol#L2
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

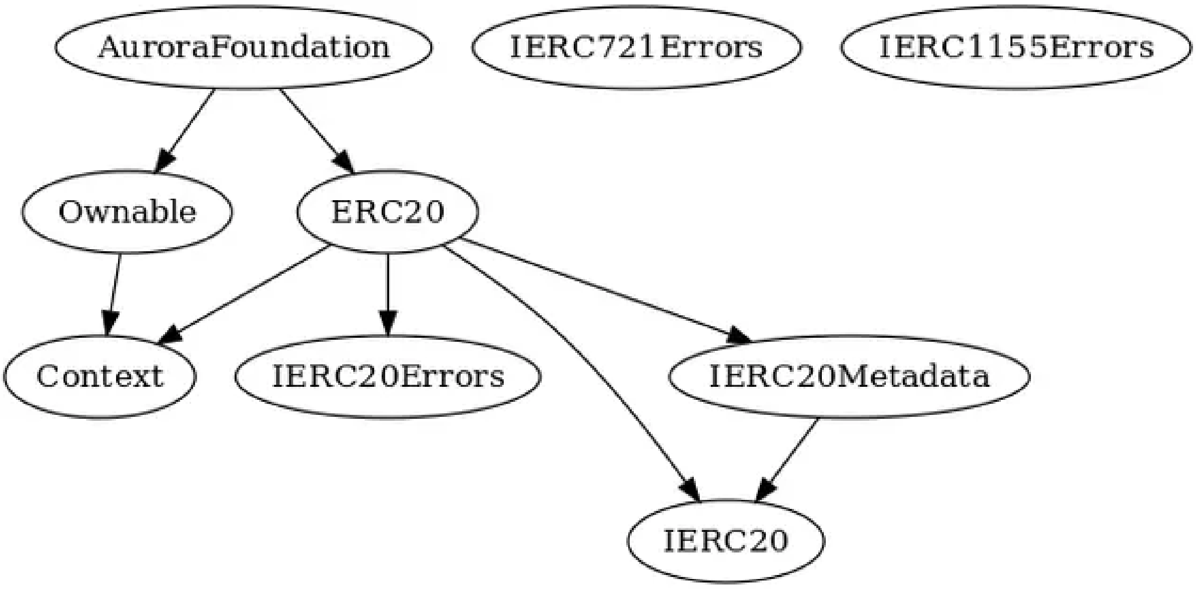
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC20Errors	Interface			
IERC721Errors	Interface			
IERC1155Errors	Interface			
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
	_contextSuffixLength	Internal		
Ownable	Implementation	Context		
		Public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

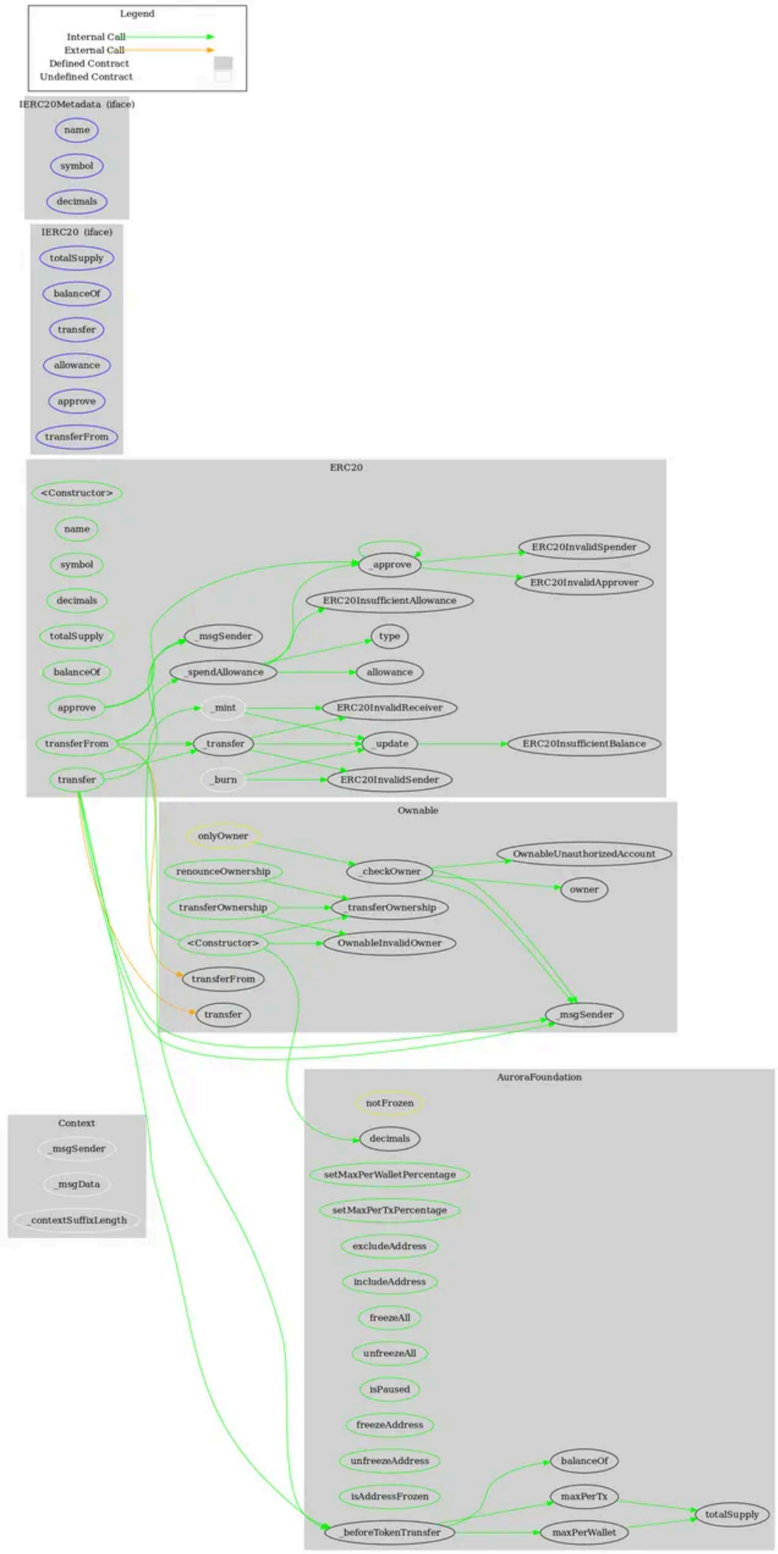
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	_transfer	Internal	✓	
	_update	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_approve	Internal	✓	
	_spendAllowance	Internal	✓	
AuroraFoundation	Implementation	ERC20, Ownable		
		Public	✓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

	freezeAll	Public	✓	onlyOwner
	unfreezeAll	Public	✓	onlyOwner
	isPaused	Public		-
	freezeAddress	Public	✓	onlyOwner
	unfreezeAddress	Public	✓	onlyOwner
	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
ERC20InvalidSpender
ERC20InvalidApprover
ERC20InsufficientAllowance
_msgSender
type
_spendAllowance
allowance
_mint
ERC20InvalidReceiver
_transfer
_update
ERC20InsufficientBalance
_burn
ERC20InvalidSender
Ownable
onlyOwner
renounceOwnership
_checkOwner
OwnableUnauthorizedAccount
owner
transferOwnership
_transferOwnership
<Constructor>
OwnableInvalidOwner
transferFrom
transfer
_msgSender
AuroraFoundation
notFrozen
decimals
setMaxPerWalletPercentage
setMaxPerTxPercentage
excludeAddress
includeAddress
freezeAll
unfreezeAll
isPaused
freezeAddress
unfreezeAddress
isAddressFrozen
balanceOf
maxPerTx
totalSupply
_beforeTokenTransfer
maxPerWallet
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. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats.

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