Cyberscope

Audit Report

Ankaa Exchange

March 2023

Table of Contents

Review

Contract Name	AnkaaToken
Testing Deploy	https://testnet.bscscan.com/address/0xc830dceccc809d34f2de4553f56a02c11a04a160
Symbol	ANKAA
Decimals	18
Total Supply	10,000,000

Audit Updates

Initial Audit	15 Mar 2023
Corrected Phase 2	17 Mar 2023

Source Files

Filename	SHA256
contracts/testingDeploy/AnkaaToken.sol	4ab9f51a7b5e60bb6f34b0193908b15e80a94eb17452b8441acda968d53d6214

Analysis

● Critical   ● Medium   ● Minor / Informative   ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	OCTD	Transfers Contract's Tokens	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	ULTW	Transfers Liquidity to Team Wallet	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

ST - Stops Transactions

Criticality	Minor / Informative
Location	contracts/AnkaaToken.sol#L1476
Status	Unresolved

Description

The contract owner has the authority to stop transactions for all users. The owner may take advantage of it by calling the `pause` function.

```solidity
function _beforeTokenTransfer(address from, address to, uint256 amount)
    internal
    whenNotPaused
    override
{
    super._beforeTokenTransfer(from, to, amount);
}
```

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. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

Diagnostics

● Critical   ● Medium   ● Minor / Informative

Severity	Code	Description	Status
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L19	Stable Compiler Version	Unresolved

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	contracts/testingDeploy/AnkaaToken.sol#L78,85,93,104,114,199,217,253,264,306, 317,355,368,398,423,448,794,803,1340
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.

```solidity
function max(uint256 a, uint256 b) internal pure returns (uint256) {
    return a > b ? a : b;
}

function min(uint256 a, uint256 b) internal pure returns (uint256) {
    return a < b ? a : b;
...
    return (a & b) + (a ^ b) / 2;
}

function ceilDiv(uint256 a, uint256 b) internal pure returns (uint256) {
    // (a + b - 1) / b can overflow on addition, so we distribute.
    return a == 0 ? 0 : (a - 1) / b + 1;
}

...
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	contracts/testingDeploy/AnkaaToken.sol#L161,164,176,180,181,182,183,184,185,191
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
denominator := div(denominator, twos)
uint256 inverse = (3 * denominator) ^ 2
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	contracts/testingDeploy/AnkaaToken.sol#L125,429
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly {
            let mm := mulmod(x, y, not(0))
            prod0 := mul(x, y)
            prod1 := sub(sub(mm, prod0), lt(mm, prod0))
        }

mbly {
            ptr := add(buffer, add(32, length))
        }
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/testingDeploy/AnkaaToken.sol#L1
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.

```solidity
pragma solidity ^0.8.9;
```

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**	
IERC165	Interface				
	supportsInterface	External		-	
ERC165	Implementation	IERC165			
	supportsInterface	Public		-	
Math	Library				
	max	Internal			
	min	Internal			
	average	Internal			
	ceilDiv	Internal			
	mulDiv	Internal			
	mulDiv	Internal			
	sqrt	Internal			
	sqrt	Internal			
	log2	Internal			
	log2	Internal			
	log10	Internal			
	log10	Internal			
	log256	Internal			
	log256	Internal			
Strings	Library				
	toString	Internal			

	toHexString	Internal		
	toHexString	Internal		
	toHexString	Internal		
IAccessControl	Interface			
	hasRole	External		-
	getRoleAdmin	External		-
	grantRole	External	✓	-
	revokeRole	External	✓	-
	renounceRole	External	✓	-
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
AccessControl	Implementation	Context, IAccessControl, ERC165		
	supportsInterface	Public		-
	hasRole	Public		-
	_checkRole	Internal		
	_checkRole	Internal		
	getRoleAdmin	Public		-
	grantRole	Public	✓	onlyRole
	revokeRole	Public	✓	onlyRole
	renounceRole	Public	✓	-
	_setupRole	Internal	✓	
	_setRoleAdmin	Internal	✓	
	_grantRole	Internal	✓	
	_revokeRole	Internal	✓	

Pausable	Implementation	Context		
		Public	✓	-
	paused	Public		-
	_requireNotPaused	Internal		
	_requirePaused	Internal		
	_pause	Internal	✓	whenNotPaused
	_unpause	Internal	✓	whenPaused
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		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-

	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_spendAllowance	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
AnkaaToken	Implementation	ERC20, Pausable, AccessControl		
		Public	✓	ERC20
	maxSupply	Public		-
	pause	Public	✓	onlyRole
	unpause	Public	✓	onlyRole
	_beforeTokenTransfer	Internal	✓	whenNotPaused

Inheritance Graph
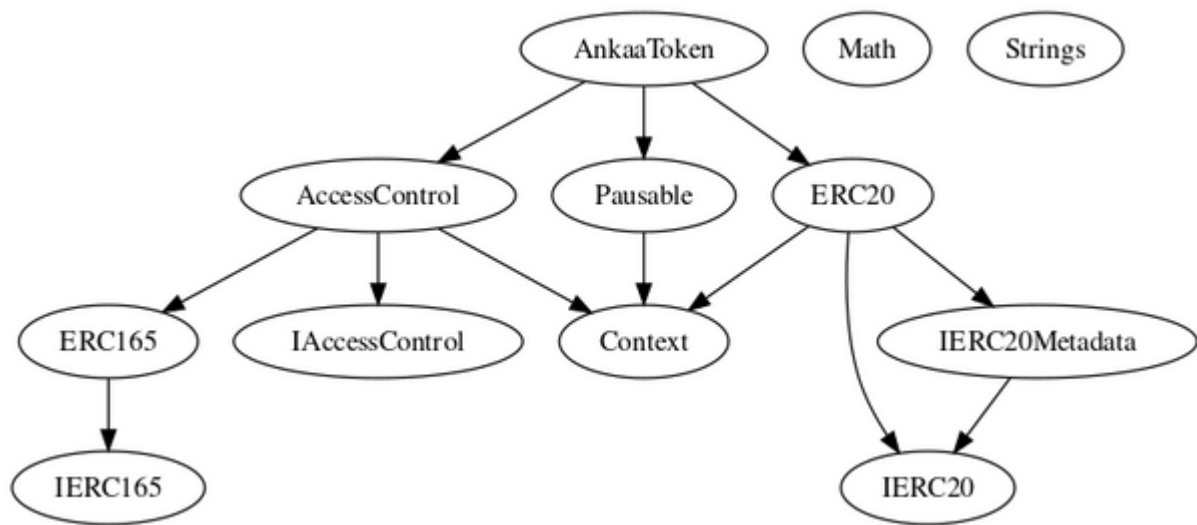
Flow Graph
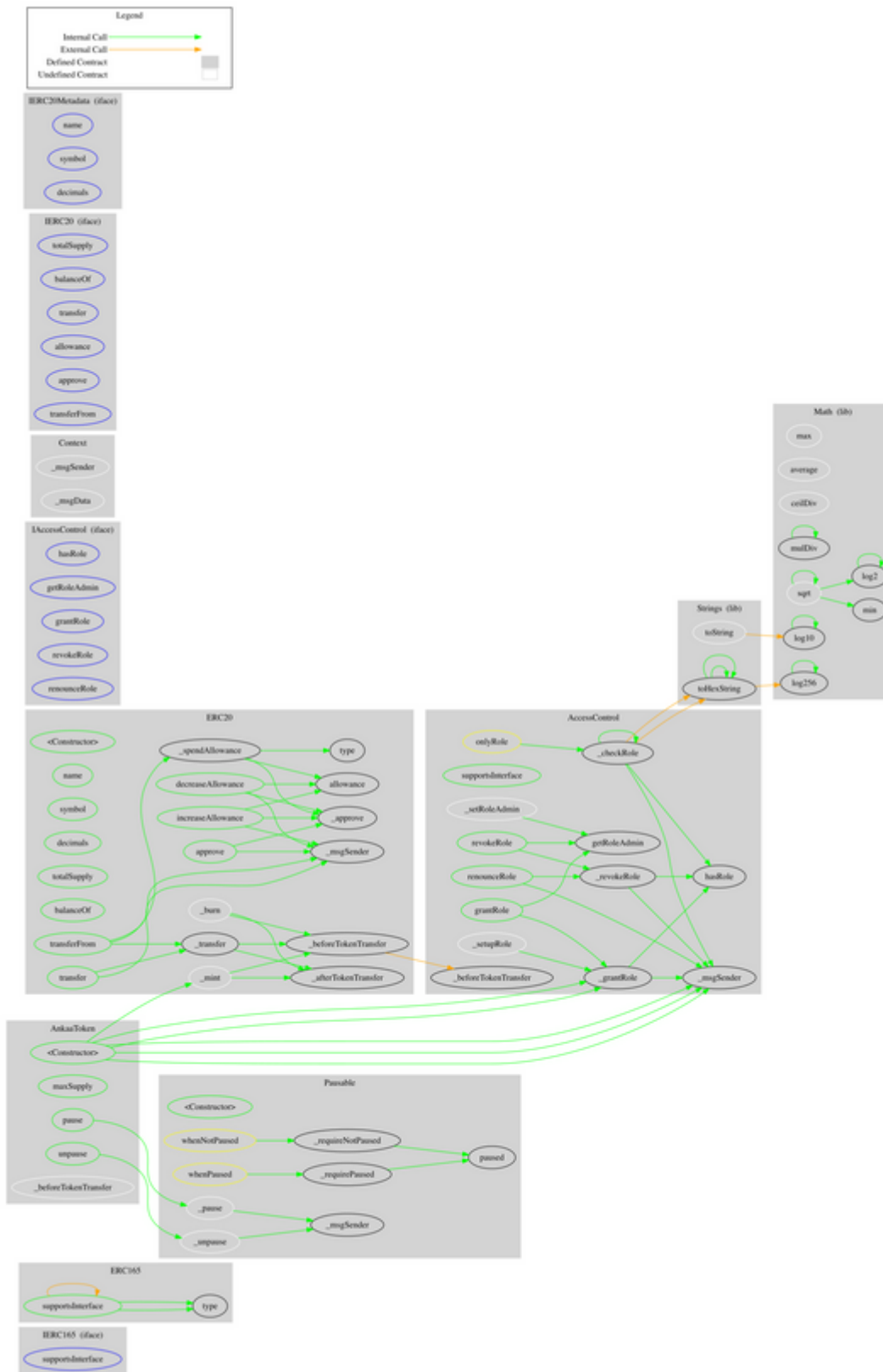
Summary

Ankaa Exchange contract implements a token mechanism. 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