Cyberscope

Audit Report

Axondao

October 2023

SHA256 45e1fba24b73082f94dbe96f3933ff793f1fd21552362294d60246a4e8f43a9b

Audited by © cyberscope

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	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
●	TSD	Total Supply Diversion	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	ZD	Zero Division	Unresolved
●	TUU	Time Units Usage	Unresolved
●	L13	Divide before Multiply Operation	Acknowledged

Table of Contents

Analysis **1**
Diagnostics **2**
Table of Contents **3**
Review **4**
Audit Updates 4
Source Files 5
Findings Breakdown **6**
TSD - Total Supply Diversion 7
Description 7
Recommendation 8
MEE - Missing Events Emission 9
Description 9
Recommendation 9
ZD - Zero Division 10
Description 10
Recommendation 10
TUU - Time Units Usage 11
Description 11
Recommendation 11
L13 - Divide before Multiply Operation 12
Description 12
Recommendation 12
Team Update 12
Functions Analysis **13**
Inheritance Graph **20**
Flow Graph **21**
Summary **22**
Disclaimer **23**
About Cyberscope **24**

Review

Contract Name	AXGT
Testing Deploy	https://testnet.bscscan.com/address/0x97cae440cf5d99a65081d9aad822ac37e40ce461
Symbol	AXGT
Decimals	18
Total Supply	1,000,000,000

Audit Updates

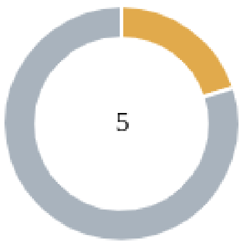
Initial Audit	20 Sep 2023 https://github.com/cyberscope-io/audits/blob/main/axgt/v1/audit.pdf
Corrected Phase 2	26 Sep 2023 https://github.com/cyberscope-io/audits/blob/main/axgt/v2/audit.pdf
Corrected Phase 3	10 Oct 2023 https://github.com/cyberscope-io/audits/blob/main/axgt/v3/audit.pdf
Corrected Phase 4	12 Oct 2023 https://github.com/cyberscope-io/audits/blob/main/axgt/v4/audit.pdf
Corrected Phase 5	13 Oct 2023 https://github.com/cyberscope-io/audits/blob/main/axgt/v5/audit.pdf

Corrected Phase 6	19 Oct 2023

Source Files

Filename	SHA256
contracts/AXGT.sol	45e1fba24b73082f94dbe96f3933ff793f1fd21552362294d60246a4e8f43a9b

Findings Breakdown



🟥	Critical	0
🟧	Medium	1
●	Minor / Informative	4

	Severity	Unresolved	Acknowledged	Resolved	Other
🟥	Critical	0	0	0	0
🟧	Medium	1	0	0	0
●	Minor / Informative	3	1	0	0

TSD - Total Supply Diversion

Criticality	Medium
Location	contracts/AXGT.sol#L772
Status	Unresolved

Description

The total supply of a token is the total number of tokens that have been created, while the balances of individual accounts represent the number of tokens that an account owns. The total supply and the balances of individual accounts are two separate concepts that are managed by different variables in a smart contract. These two entities should be equal to each other.

In the contract, the amount that is added to the total supply does not equal the amount that is added to the balances. As a result, the sum of balances is diverse from the total supply.

Specifically, the contract allows the `onlyVestDistributer` to distribute vested tokens to another account using the `distributeVest` function. This function transfers tokens and sets a `lockedVest` amount for the recipient. However, when invoking the balanceOf for an account, the function subtracts the `lockedVest` amount from the `_balances[account]`. As a result, the `balanceOf` function will return a value that is lower than the actaul `totalSupply` variable. This is because the vested amount, calculated by the `lockedVest`, is subtracted, resulting in smaller balance. Furthermore, if the owner burns tokens such that their `_balances` becomes less than the `lockedVest` amount, the `balanceOf` function for the owner's account will underflow.

```
 function _burnToken (uint256 _amount) external onlyOwner{
        _burn(owner(), _amount);
    }

    function balanceOf(address account) public view override returns
(uint256) {
        return _balances[account]- lockedVest(account);
    }

    function distributeVest(
        uint256 _numberOfVestedMonths,
        uint256 amount,
        address holder
    ) external onlyVestDistributer {
        require(_vestAmount[holder] == 0, "Vest already granted");

        super._transfer(vestDistributer, holder, amount);

        numberOfVestedMonths[holder] = _numberOfVestedMonths;
        _vestAmount[holder] = amount;
        emit VestGranted(holder, amount);
    }
```

Recommendation

The total supply and the balance variables are separate and independent from each other. The total supply represents the total number of tokens that have been created, while the balance mapping stores the number of tokens that each account owns. The sum of balances should always equal the total supply.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	contracts/AXGT.sol#L921
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 setVestDistributer(address _vestDistributer) external
onlyOwner {
        vestDistributer = _vestDistributer;
    }
```

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.

ZD - Zero Division

Criticality	Minor / Informative
Location	contracts/AXGT.sol#L775,852
Status	Unresolved

Description

The contract is using variables that may be set to zero as denominators. This can lead to unpredictable and potentially harmful results, such as a transaction revert.

Specifically, the `onlyVestDistributer` can invoke the `distributeVest` and set the `numberOfVestedMonths` variable to zero.

```
    function distributeVest(
        uint256 _numberOfVestedMonths,
        uint256 amount,
        address holder
    ) external onlyVestDistributer {
        require(_vestAmount[holder] == 0, "Vest already granted");

        super._transfer(vestDistributer, holder, amount);

        numberOfVestedMonths[holder] = _numberOfVestedMonths;
        _vestAmount[holder] = amount;
        emit VestGranted(holder, amount);
    }
....
uint256 unlockedVest = (_vestAmount[account] * passedMonths) /
numberOfVestedMonths[account];
```

Recommendation

It is important to handle division by zero appropriately in the code to avoid unintended behavior and to ensure the reliability and safety of the contract. The contract should ensure that the divisor is always non-zero before performing a division operation. It should prevent the variables to be set to zero, or should not allow the execution of the corresponding statements.

TUU - Time Units Usage

Criticality	Minor / Informative
Location	contracts/AXGT.sol#L684
Status	Unresolved

Description

The contract is using arbitrary numbers to form time-related values. As a result, it decreases the readability of the codebase and prevents the compiler to optimize the source code.

```
uint256 private constant ONE_MONTH = 2592000;
uint256 private constant MARS2024 = 1709251200;
```

Recommendation

It is a good practice to use the time units reserved keywords like `seconds`, `minutes`, `hours`, `days` and `weeks` to process time-related calculations.

It's important to note that these time units are simply a shorthand notation for representing time in seconds, and do not have any effect on the actual passage of time or the execution of the contract. The time units are simply a convenience for expressing time in a more human-readable form.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	contracts/AXGT.sol#L836,843,848,852
Status	Acknowledged

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.

```
uint256 passedMonths = (block.timestamp - MARS2024) /
ONE_MONTH;

uint256 unlockedVest = (_vestAmount[account] * passedMonths) /
numberOfVestedMonths[account];
```

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.

Team Update

The team has acknowledged that this is not a security issue and states:

The vest is being released in a monthly basis so the rest of the division (number of seconds in the next non completed month) should be ignored.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
ReentrancyGuard	Implementation			
Context	Implementation			
	_msgSender	Internal		
	_msgData	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		
		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	✓	
	_beforeTokenTransfer	Internal	✓	
Ownable	Implementation	Context		
		Public	✓	-

	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
IUniswapV2Pair	Interface			
	name	External		-
	symbol	External		-
	decimals	External		-
	totalSupply	External		-
	balanceOf	External		-
	allowance	External		-
	approve	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	DOMAIN_SEPARATOR	External		-
	PERMIT_TYPEHASH	External		-
	nonces	External		-
	permit	External	✓	-
	MINIMUM_LIQUIDITY	External		-
	factory	External		-
	token0	External		-
	token1	External		-
	getReserves	External		-

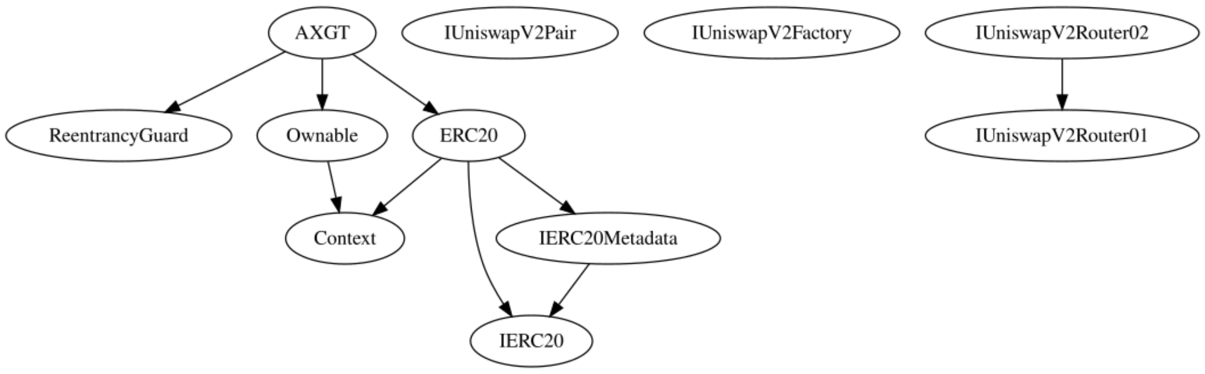
	price0CumulativeLast	External		-
	price1CumulativeLast	External		-
	kLast	External		-
	mint	External	✓	-
	burn	External	✓	-
	swap	External	✓	-
	skim	External	✓	-
	sync	External	✓	-
	initialize	External	✓	-
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-

	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-
	swapTokensForExactETH	External	✓	-
	swapExactTokensForETH	External	✓	-
	swapETHForExactTokens	External	Payable	-
	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-
	getAmountsOut	External		-
	getAmountsIn	External		-
IUniswapV2Router02	Interface	IUniswapV2Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-

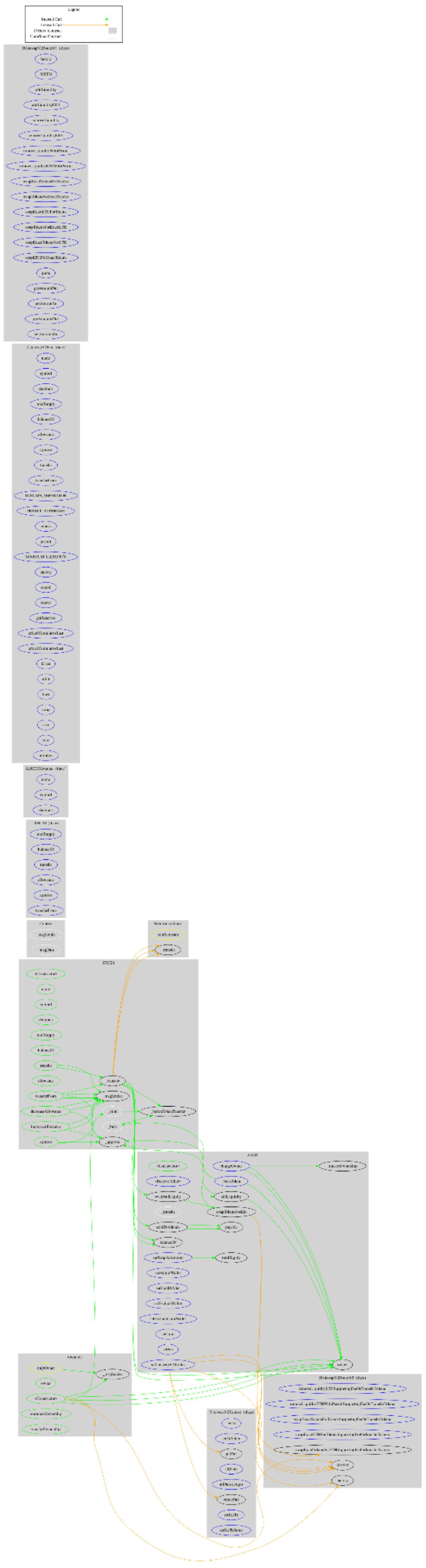
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
AXGT	Implementation	ERC20, Ownable, ReentrancyGuard		
		Public	✓	ERC20
		External	Payable	-
	setUniswapV2Router	External	✓	onlyOwner
	_burnToken	External	✓	onlyOwner
	balanceOf	Public		-
	distributeVest	External	✓	onlyVestDistributer
	_transfer	Internal	✓	
	swapAndLiquify	Private	✓	
	lockedVest	Public		-
	swapTokensForEth	Private	✓	
	addLiquidity	Private	✓	
	sendDividends	Private	✓	nonReentrant
	setSwapAtAmount	External	✓	onlyOwner
	setAdminWallet	External	✓	onlyOwner
	setFundWAllet	External	✓	onlyOwner
	setVestDistributer	External	✓	onlyOwner
	changeOwner	External	✓	onlyOwner
	setExcludeWallet	External	✓	onlyOwner

	setExcludeLimitWallet	External	✓	onlyOwner
	setLimit	External	✓	onlyOwner
	setFee	External	✓	onlyOwner

Inheritance Graph



Flow Graph



Summary

Axondao contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. Axondao 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 25% fees.

Disclaimer

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