Cyberscope

Audit Report
Archon

January 2023

Type	BEP20
Network	BSC
Address	0x8A9a97591503515538CD167E0cF05BE7C004628C
Audited by	© cyberscope

Table of Contents

Review

Contract Name	Archon
Compiler Version	v0.8.7+commit.e28d00a7
Optimization	200 runs
Explorer	https://bscscan.com/address/0x8a9a97591503515538cd167e0cf05be7c004628c
Address	0x8a9a97591503515538cd167e0cf05be7c004628c
Network	BSC
Symbol	$ARCH
Decimals	9
Total Supply	10.000.000.000

Audit Updates

Initial Audit	20 Jan 2023

Source Files

Filename	SHA256
Archon.sol	3be75295b1093b43597640c80e32b8b2f9714e6730b77dc1104705dc19c21bcd

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

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

ULTW - Transfers Liquidity to Team Wallet

Criticality	Minor / Informative
Status	Unresolved

Description

The contract owner has the authority to transfer funds without limit to the team wallet. These funds have been accumulated from fees collected from the contract. The owner may take advantage of it by calling the `sweepContingency` method.

```
function sweepContingency() external onlyOwner {
  payable(owner()).transfer(address(this).balance);
}
```

Recommendation

The contract could embody a check for the maximum amount of funds that can be swapped. Since a huge amount may volatile the token's price. 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. That risk can be prevented by temporarily locking the contract or renouncing ownership.

OCTD - Transfers Contract's Tokens

Criticality	Minor / Informative
Location	Archon.sol#L345,616
Status	Unresolved

Description

The contract owner has the authority to claim all the balance of the contract. The owner may take advantage of it by calling the `withdrawBUSD` and `claimTax` methods.

```
function withdrawBUSD(address tAddress, uint amount, uint tDecimals)
public onlyOwner
{
  token= IERC20(tAddress);
  token.transfer(msg.sender,amount*10**tDecimals);
}

function claimTax(address receiver, uint tAmount) public onlyOwner
{
    require(balanceOf(address(this))>=tAmount*10**_decimals,"Contract
balance is low. Refill and transfer");
    _transfer(address(this),receiver, tAmount*10**_decimals);
}
```

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. That risk can be prevented by temporarily locking the contract or renouncing ownership.

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	PVC	Price Volatility Concern	Unresolved
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	BLC	Business Logic Concern	Unresolved
●	UI	Unverified Interface	Unresolved
●	MC	Missing Check	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L11	Unnecessary Boolean equality	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	Archon.sol#L535,606
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `swapTokensAtAmount` sets a threshold where the contract will trigger the swap functionality. If the variable is set to a big number, then the contract will swap a huge amount of tokens for ETH.

It is important to note that the price of the token representing it, can be highly volatile. This means that the value of a price volatility swap involving Ether could fluctuate significantly at the triggered point, potentially leading to significant price volatility for the parties involved.

```
function setSwapSettings(uint256 thresholdPercent, uint256
thresholdDivisor, uint256 amountPercent, uint256 amountDivisor, uint256
intervalInSeconds) external onlyOwner {
    swapThreshold = (_tTotal * thresholdPercent) / thresholdDivisor;
    swapAmount = (_tTotal * amountPercent) / amountDivisor;
    swapInterval = intervalInSeconds;
}
    uint256 contractTokenBalance = balanceOf(address(this));
    if (contractTokenBalance >= swapThreshold && lastSwap +
swapInterval < block.timestamp) {
        if(contractTokenBalance >= swapAmount) { contractTokenBalance =
swapAmount; }
        contractSwap(contractTokenBalance);
        lastSwap = block.timestamp;
    }
```

Recommendation

The contract could ensure that it will not sell more than a reasonable amount of tokens in a single transaction. A suggested implementation could check that the maximum amount should be less than a fixed percentage of the total supply. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Status	Unresolved

Description

The contract sends funds to a `marketingWallet` as part of the transfer flow. This address can either be a wallet address or a contract. If the address is a contract then it may revert from incoming payment. As a result, the error will propagate to the token's contract and revert the transfer.

```
if (address(this).balance > 0 && _ratios.total - _ratios.liquidity > 0)
{
    uint256 amountBNB = address(this).balance;
    _taxWallets.development.transfer((amountBNB * _ratios.development)
/ (_ratios.total - _ratios.liquidity));
    _taxWallets.marketing.transfer(address(this).balance);
}
```

Recommendation

The contract should tolerate the potential revert from the underlying contracts when the interaction is part of the main transfer flow. This could be archived by not allowing set contract addresses or by sending the funds in a non-revertable way.

BLC - Business Logic Concern

Criticality	Minor / Informative
Location	Archon.sol#L590,482,486,490
Status	Unresolved

Description

The implementation may not follow the expected behavior. The contract contains an if statement that appears to be unnecessary.

```
if(to != currentRouter && !lpPairs[to]) {
    //require(balanceOf(to) + amount <= (_maxWalletSizePercent *
getCirculatingSupply()) / 1000, "Transfer amount exceeds the
maxWalletSize.");
}
```

The contract has an AntiSnipe Interface that can be used to blacklist users, but it is not being applied to the transactions. Therefore, it is redundant.

```
function removeBlacklisted(address account) external onlyOwner {
    antiSnipe.removeBlacklisted(account);
}

function isBlacklisted(address account) public view returns (bool) {
    return antiSnipe.isBlacklisted(account);
}

function getSniperAmt() public view returns (uint256) {
    return antiSnipe.getSniperAmt();
}
```

The contract verifies that the sender has enough funds to complete the transaction, but it transfers different transfer amount `amounts[i]*10**_decimals` to the receiver.

```
function multiSendTokens(address[] memory accounts, uint256[] memory
amounts) external {
    require(accounts.length == amounts.length, "Lengths do not
match.");
    for (uint8 i = 0; i < accounts.length; i++) {
        require(balanceOf(msg.sender) >= amounts[i]);
        _transfer(msg.sender, accounts[i], amounts[i]*10**_decimals);
    }
}
```

Recommendation

The team is advised to carefully check if the implementation follows the expected business logic. It is recommended to remove the unnecessary code statement from the contract, as it serves no purpose and may cause confusion or errors. Unnecessary code can also make it more difficult to understand and maintain the contract.

UI - Unverified Interface

Criticality	Minor / Informative
Location	Archon.sol#L287,476
Status	Unresolved

Description

The contract uses an untrusted external contract.

```
AntiSnipe antiSnipe;

function setInitializer(address initializer) external onlyOwner {
    require(!_hasLiqBeenAdded, "Liquidity is already in.");
    require(initializer != address(this), "Can't be self.");
    antiSnipe = AntiSnipe(initializer);
}
```

Recommendation

The contract should use a trusted external source. A trusted source could be either a commonly recognized or an audited contract.

MC - Missing Check

Criticality	Minor / Informative
Location	Archon.sol#L432,445
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues.The `newRouter` , `pair` ,and `router` addresses can be set to zero address.

```
function setNewRouter(address newRouter) public onlyOwner() {
    IRouter02 _newRouter = IRouter02(newRouter);
    address get_pair =
IFactoryV2(_newRouter.factory()).getPair(address(this),
_newRouter.WETH());
    if (get_pair == address(0)) {
        lpPair =
IFactoryV2(_newRouter.factory()).createPair(address(this),
_newRouter.WETH());
    }
    else {
        lpPair = get_pair;
    }
    dexRouter = _newRouter;
    _approve(address(this), address(dexRouter), type(uint256).max);
}
function setLpPair(address pair, bool enabled) external onlyOwner {
    if (enabled == false) {
        lpPairs[pair] = false;
        antiSnipe.setLpPair(pair, false);
    } else {
        if (timeSinceLastPair != 0) {
            require(block.timestamp - timeSinceLastPair > 3 days, "3
Day cooldown.!");
        }
        lpPairs[pair] = true;
        timeSinceLastPair = block.timestamp;
        antiSnipe.setLpPair(pair, true);
    }
}
function changeRouterContingency(address router) external onlyOwner {
    require(!_hasLiqBeenAdded);
    currentRouter = router;
}
```

The nominator and denominator of the following percentages are not being properly sanitized.

```
function setSwapSettings(uint256 thresholdPercent, uint256
thresholdDivisor, uint256 amountPercent, uint256 amountDivisor, uint256
intervalInSeconds) external onlyOwner {
    swapThreshold = (_tTotal * thresholdPercent) / thresholdDivisor;
    swapAmount = (_tTotal * amountPercent) / amountDivisor;
    swapInterval = intervalInSeconds;
}
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

- The address arguments should not be set to zero address.
- It is important to ensure that the nominator is less than or equal to the denominator, in order to avoid any inaccuracies or errors in the calculated percentage.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	Archon.sol#L202,205,209
Status	Unresolved

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease the gas consumption of the corresponding transaction.

```
bool private allowedPresaleExclusion = true
uint256 private startingSupply = 10_000_000_000
uint8 private _decimals = 9
```

Recommendation

Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	Archon.sol#L140,207,208,233,239,265,282,494,543,617
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 WETH() external pure returns (address);
string constant private _name = "Archon"
string constant private _symbol = "$ARCH"

Fees public _taxRates = Fees({
        buyFee: 200,
        sellFee: 400,
...

Ratios public _ratios = Ratios({
        liquidity: 2,
        marketing: 1,
        development: 1,
        total: 4
        })

...
```

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.

L05 - Unused State Variable

Criticality	Minor / Informative
Location	Archon.sol#L199
Status	Unresolved

Description

An unused state variable is a state variable that is declared in the contract, but is never used in any of the contract's functions. This can happen if the state variable was originally intended to be used, but was later removed or never used.

Unused state variables can create clutter in the contract and make it more difficult to understand and maintain. They can also increase the size of the contract and the cost of deploying and interacting with it.

```
address[] private _excluded
```

Recommendation

To avoid creating unused state variables, it's important to carefully consider the state variables that are needed for the contract's functionality, and to remove any that are no longer needed. This can help improve the clarity and efficiency of the contract.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	Archon.sol#L523,532
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.

```
_maxTxAmountPercent = percent
swapThreshold = (_tTotal * thresholdPercent) / thresholdDivisor
```

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	Archon.sol#L667
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 _checkLiquidityAdd(address from, address to) private {
        require(!_hasLiqBeenAdded, "Liquidity already added and
marked.");
        if (!_hasLimits(from, to) && to == lpPair) {
            _liquidityHolders[from] = true;
            _hasLiqBeenAdded = true;
            if(address(antiSnipe) == address(0)){
                antiSnipe = AntiSnipe(address(this));
            }
            contractSwapEnabled = true;
            emit ContractSwapEnabledUpdated(true);
        }
    }
```

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	Archon.sol#L442
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.

```
enabled == 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	Archon.sol#L457
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.

```
currentRouter = router
```

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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	Archon.sol#L343
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
token.transfer(msg.sender,amount*10**tDecimals)
```

Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

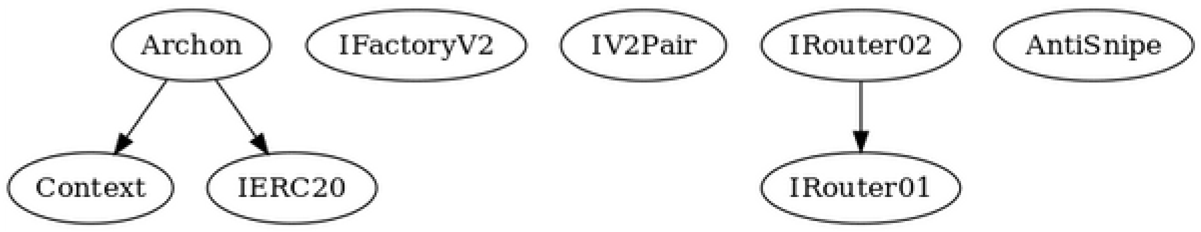
Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
IERC20	Interface			
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IFactoryV2	Interface			
	getPair	External		-
	createPair	External	✓	-
IV2Pair	Interface			
	factory	External		-
	getReserves	External		-

IRouter01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidityETH	External	Payable	-
	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-
	getAmountsOut	External		-
	getAmountsIn	External		-
IRouter02	Interface	IRouter01		
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
AntiSnipe	Interface			
	checkUser	External	✓	-
	setLaunch	External	✓	-
	setLpPair	External	✓	-
	setProtections	External	✓	-
	setGasPriceLimit	External	✓	-
	removeSniper	External	✓	-
	getSniperAmt	External		-
	removeBlacklisted	External	✓	-
	isBlacklisted	External		-
	transfer	External	✓	-
Archon	Implementation	Context, IERC20		
		Public	Payable	-

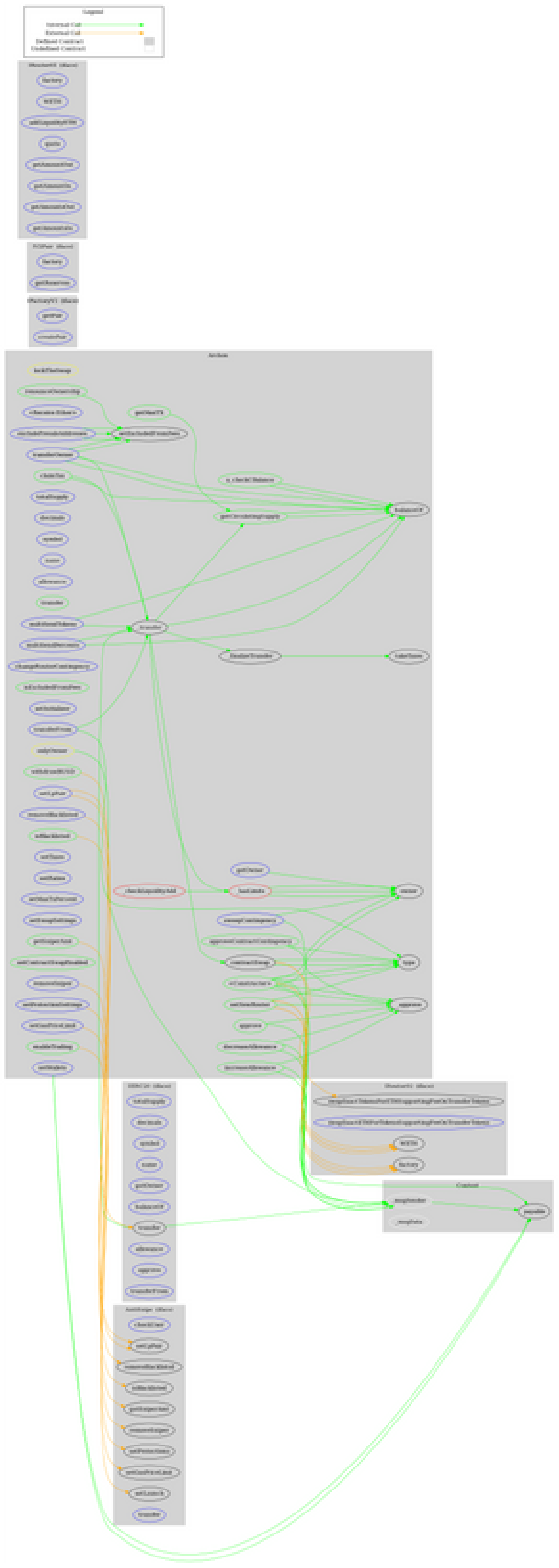
		External	Payable	-
	withdrawBUSD	Public	✓	onlyOwner
	owner	Public		-
	transferOwner	External	✓	onlyOwner
	renounceOwnership	Public	✓	onlyOwner
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	allowance	External		-
	balanceOf	Public		-
	transfer	Public	✓	-
	approve	Public	✓	-
	_approve	Private	✓	
	approveContractContingency	Public	✓	onlyOwner
	transferFrom	External	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	setNewRouter	Public	✓	onlyOwner
	setLpPair	External	✓	onlyOwner
	changeRouterContingency	External	✓	onlyOwner
	getCirculatingSupply	Public		-
	isExcludedFromFees	Public		-
	setExcludedFromFees	Public	✓	onlyOwner
	setInitializer	External	✓	onlyOwner
	removeBlacklisted	External	✓	onlyOwner
	isBlacklisted	Public		-
	getSniperAmt	Public		-

	removeSniper	External	✓	onlyOwner
	setProtectionSettings	External	✓	onlyOwner
	setGasPriceLimit	External	✓	onlyOwner
	setTaxes	External	✓	onlyOwner
	setRatios	External	✓	onlyOwner
	setMaxTxPercent	External	✓	onlyOwner
	getMaxTX	Public		-
	setSwapSettings	External	✓	onlyOwner
	setWallets	External	✓	onlyOwner
	setContractSwapEnabled	Public	✓	onlyOwner
	excludePresaleAddresses	External	✓	onlyOwner
	_hasLimits	Private		
	_transfer	Internal	✓	
	claimTax	Public	✓	onlyOwner
	a_checkCBalance	Public	✓	onlyOwner
	contractSwap	Private	✓	lockTheSwap
	_checkLiquidityAdd	Private	✓	
	enableTrading	Public	✓	onlyOwner
	sweepContingency	External	✓	onlyOwner
	multiSendTokens	External	✓	-
	multiSendPercents	External	✓	-
	takeTaxes	Internal	✓	
	_finalizeTransfer	Private	✓	

Inheritance Graph



Flow Graph



Summary

There are some functions that can be abused by the owner like draining the contract's tokens and transferring funds to the team's wallet. 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

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