Audit Report

VOLTRON

May 2023

Network	BSC
Address	0x8647b1c35e623f934d7ea2469d4cf5fce1b40985
Audited by	© cyberscope

Table of Contents

Review

Contract Name	VOLTRONTOKEN
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	200 runs
Explorer	https://bscscan.com/address/0x8647b1c35e623f934d7ea2469d4cf5fce1b40985
Address	0x8647b1c35e623f934d7ea2469d4cf5fce1b40985
Network	BSC
Symbol	VOLTN
Decimals	18
Total Supply	100,000,000,000,000

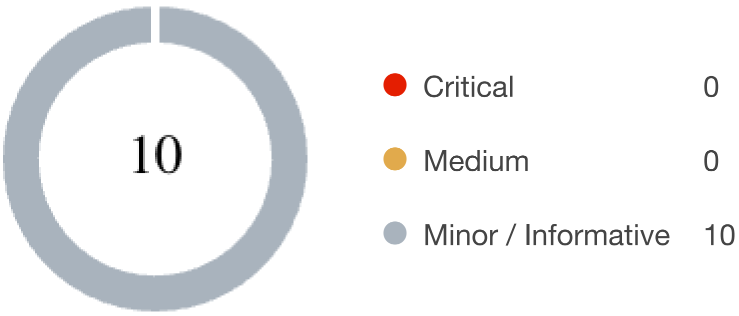
Audit Updates

Initial Audit	14 May 2023
Corrected Phase 2	19 May 2023
Corrected Phase 3	21 May 2023

Source Files

Filename	SHA256
VOLTRONTOKEN.sol	5bcb334abb40afd7efa2a90ae3b8bb42b2c4fcfcb34d89dd8c231b799a18fbcf

Findings Breakdown



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

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	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

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	DDP	Decimal Division Precision	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	RSR	Redundant Swap Repetition	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	contracts/testingDeploy/voltron.sol#L846
Status	Unresolved

Description

Division of decimal (fixed point) numbers can result in rounding errors due to the way that division is implemented in Solidity. Thus, it may produce issues with precise calculations with decimal numbers.

Solidity represents decimal numbers as integers, with the decimal point implied by the number of decimal places specified in the type (e.g. decimal with 18 decimal places). When a division is performed with decimal numbers, the result is also represented as an integer, with the decimal point implied by the number of decimal places in the type. This can lead to rounding errors, as the result may not be able to be accurately represented as an integer with the specified number of decimal places.

Hence, the splitted shares will not have the exact precision and some funds may not be calculated as expected.

```
if (marketingShare > 0) {
    uint256 marketingTokens = contractTokenBalance * marketingShare
/ totalFee;
    swapAndSendMarketing(marketingTokens);
}

if (developmentShare > 0) {
    uint256 developmentTokens = contractTokenBalance *
developmentShare / totalFee;
    swapAndSendDevelopment(developmentTokens);
}
```

Recommendation

The contract could calculate the subtraction of the divided funds in the last calculation in order to avoid the division rounding issue.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	contracts/testingDeploy/voltron.sol#L879
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `setSwapTokensAtAmount` 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 setSwapTokensAtAmount(uint256 newAmount) external
onlyOwner{
    require(newAmount > totalSupply() / 1_000_000,
"SwapTokensAtAmount must be greater than 0.0001% of total supply");
    swapTokensAtAmount = newAmount;

    emit SwapTokensAtAmountUpdated(swapTokensAtAmount);
}
```

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
Location	contracts/testingDeploy/voltron.sol#L901,921
Status	Unresolved

Description

The contract sends funds to `marketingWallet` and `developmentWallet` as part of the transfer flow. This address can either be a wallet address or a contract. If the address belongs to 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.

```
payable(marketingWallet).sendValue(newBalance);
payable(developmentWallet).sendValue(newBalance);
```

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 achieved by not allowing set contract addresses or by sending the funds in a non-revertable way.

RSR - Redundant Swap Repetition

Criticality	Minor / Informative
Location	contracts/testingDeploy/voltron.sol#L833
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.

As part of the swap flow, the contract swap and transfer native tokens twice. One for marketing and one for the development wallet. The repetition of the swap process increases uneseccarily the gas cost.

```
swapAndSendMarketing(marketingTokens);
// ...
swapAndSendDevelopment(developmentTokens);
```

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.

The contract could swap the total amount of marketing and development shares once and transfer the propotional funds to the corresponding address.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	VOLTRONTOKEN.sol#L733,734,740
Status	Unresolved

Description

The contract is using variables that initialize them only in the constructor. The other functions are not mutating the variables. These variables are not defined as `immutable`.

```
uniswapV2Router
uniswapV2Pair
maxFee
```

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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	VOLTRONTOKEN.sol#L59,60,77,97,709,777,788,800,807
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 DOMAIN_SEPARATOR() external view returns (bytes32);
function PERMIT_TYPEHASH() external pure returns (bytes32);
function MINIMUM_LIQUIDITY() external pure returns (uint);
function WETH() external pure returns (address);
event developmentWalletChanged(address developmentWallet);
uint256 _marketingFeeOnBuy
uint256 _developmentFeeOnBuy
uint256 _developmentFeeOnSell
uint256 _marketingFeeOnSell
address _marketingWallet
address _developmentWallet
```

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	VOLTRONTOKEN.sol#L279,328,338,357,371,388,398,413,423,438,462,474,648
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 isContract(address account) internal view returns (bool) {
        // This method relies on extcodesize/address.code.length,
which returns 0
        // for contracts in construction, since the code is only
stored at the end
        // of the constructor execution.

        return account.code.length > 0;
    }

function functionCall(address target, bytes memory data) internal
returns (bytes memory) {
        return functionCallWithValue(target, data, 0, "Address:
low-level call failed");
    }

...
```

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.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	VOLTRONTOKEN.sol#L719
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
address router
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	VOLTRONTOKEN.sol#L479
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 returndata_size := mload(returndata)
                revert(add(32, returndata), returndata_size)
            }
```

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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	VOLTRONTOKEN.sol#L766
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.

```
ERC20token.transfer(msg.sender, balance)
```

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.

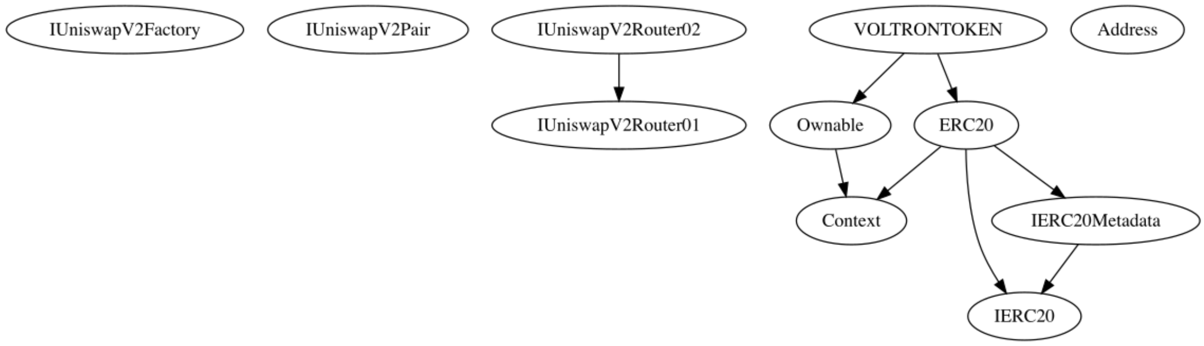
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	verifyCallResultFromTarget	Internal		
	verifyCallResult	Internal		
	_revert	Private		
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		

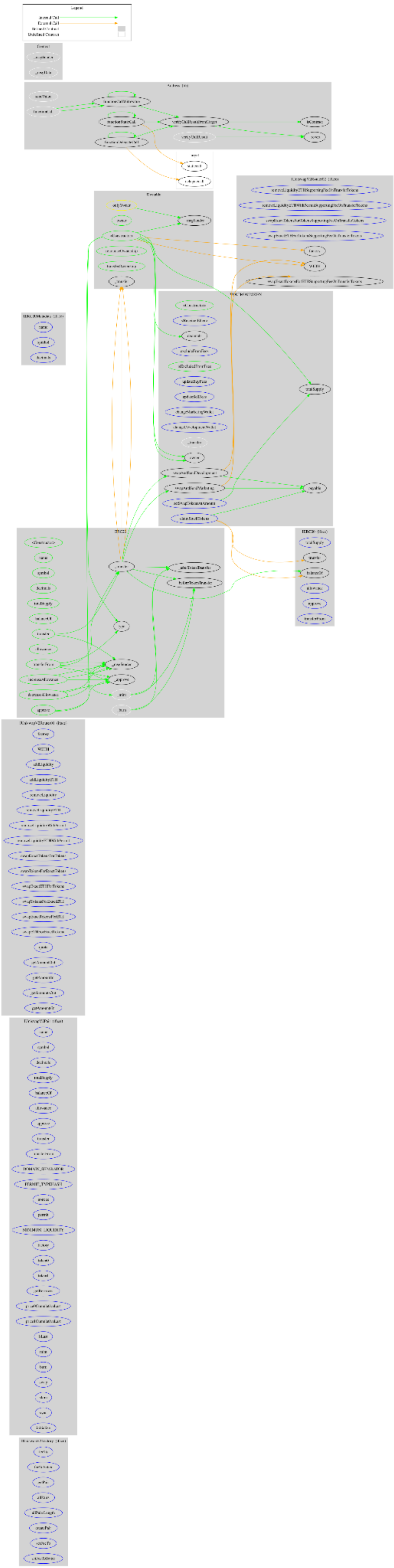
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	

	_approve	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
VOLTRONTOKEN	Implementation	ERC20, Ownable		
		Public	✓	ERC20
		External	Payable	-
	claimStuckTokens	External	✓	onlyOwner
	excludeFromFees	External	✓	onlyOwner
	isExcludedFromFees	Public		-
	updateBuyFees	External	✓	onlyOwner
	updateSellFees	External	✓	onlyOwner
	changeMarketingWallet	External	✓	onlyOwner
	changeDevelopmentWallet	External	✓	onlyOwner
	_transfer	Internal	✓	
	setSwapTokensAtAmount	External	✓	onlyOwner
	swapAndSendMarketing	Private	✓	
	swapAndSendDevelopment	Private	✓	

Inheritance Graph



Flow Graph



Summary

VOLTRON contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. VOLTRON 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

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