Cyberscope

Audit Report

METALBANK X

February 2025

Network	ETH
Address	0x00705c3a7a59cebc5214777777e58dd74b27f669
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
Critical	TSD	Total Supply Diversion	Unresolved
Minor / Informative	CCR	Contract Centralization Risk	Unresolved
Minor / Informative	IDI	Immutable Declaration Improvement	Unresolved
Minor / Informative	MTEE	Missing Transfer Event Emission	Unresolved
Minor / Informative	NWES	Nonconformity with ERC-20 Standard	Unresolved
Minor / Informative	RBTS	Resetting Balance and Total Supply	Unresolved
Minor / Informative	TFPC	Token Fixed Price Concern	Unresolved
Minor / Informative	UCM	Unnecessary Comment Messages	Unresolved
Minor / Informative	L02	State Variables could be Declared Constant	Unresolved
Minor / Informative	L04	Conformance to Solidity Naming Conventions	Unresolved
Minor / Informative	L16	Validate Variable Setters	Unresolved
Minor / Informative	L19	Stable Compiler Version	Unresolved

Table of Contents

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
Critical	Highly Likely / High Impact
Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
Minor / Informative	Unlikely / Low to no Impact

Review

Contract Name	HashnodeTestCoin
Compiler Version	v0.8.28+commit.7893614a
Optimization	200 runs
Explorer	https://etherscan.io/address/0x00705c3a7a59cebc5214777777e58dd74b27f669
Address	0x00705c3a7a59cebc5214777777e58dd74b27f669
Network	ETH
Symbol	MBXAU
Decimals	18
Total Supply	1.000.000.000
Badge Eligibility	Must Fix Criticals

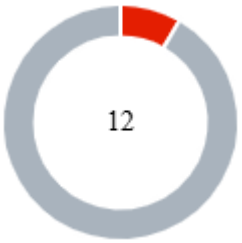
Audit Updates

Initial Audit	13 Feb 2025 https://github.com/cyberscope-io/audits/blob/main/mbxau/v1/audit.pdf
Corrected Phase 2	26 Feb 2025

Source Files

Filename	SHA256
HashnodeTestCoin.sol	f338328c10544c265f265612b38c7451628d28824bb3ff2e7a348420358db1a6

Findings Breakdown



Critical	1
Medium	0
Minor / Informative	11

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	0	0	0	0
Minor / Informative	11	0	0	0

TSD - Total Supply Diversion

Criticality	Critical
Location	HashnodeTestCoin.sol#L66,67
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.

```
_totalSupply = 10000000000000000000000000;
balances[msg.sender] = 100000000000000000000000000000;

// total supply -> 1_000_000_000_000_000_000_000_000_000
// balance      -> 1_000_000_000_000_000_000_000_000_000_000
```

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.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	HashnodeTestCoin.sol#L67,80
Status	Unresolved

Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

Specifically, in the `receive` function, users pay an amount of ETH in order to buy tokens. The tokens are kept in the address of the `fundsWallet` however the `fundsWallet` could transfer all the funds into another account resulting in users not being able to buy tokens.

```
constructor(uint256 initialSupply) StandardToken(initialSupply)
{
    _totalSupply = 100000000000000000000000000;
    balances[msg.sender] = 100000000000000000000000000000000;
    //...
    fundsWallet = msg.sender;
}

receive() external payable {
    //...
    require(balances[fundsWallet] >= amount);
    balances[fundsWallet] -= amount;
    balances[msg.sender] += amount;
    //...
}
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	HashnodeTestCoin.sol#L69,71,72
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.

```
decimals
unitsOneEthCanBuy
fundsWallet
```

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.

MTEE - Missing Transfer Event Emission

Criticality	Minor / Informative
Location	HashnodeTestCoin.sol#L54,67
Status	Unresolved

Description

The contract does not emit an event when portions of the main amount are transferred during the transfer process. This lack of event emission results in decreased transparency and traceability regarding the flow of tokens, and hinders the ability of decentralized applications (dApps), such as blockchain explorers, to accurately track and analyze these transactions.

```
    constructor(uint256 initialSupply) {
        _totalSupply = initialSupply;
        balances[msg.sender] = _totalSupply;
    }
    //...
    constructor(uint256 initialSupply)
StandardToken(initialSupply) {
        //...
        balances[msg.sender] = 10000000000000000000000000000000;
        //...
    }
```

Recommendation

It is advisable to incorporate the emission of detailed event logs following each asset transfer. These logs should encapsulate key transaction details, including the identities of the sender and receiver, and the quantity of assets transferred. Implementing this practice will enhance the reliability and transparency of transaction tracking systems, ensuring accurate data availability for ecosystem participants.

NWES - Nonconformity with ERC-20 Standard

Criticality	Minor / Informative
Location	HashnodeTestCoin.sol#L22,30
Status	Unresolved

Description

The contract does not fully conform to the ERC20 Standard. Specifically, according to the standard, transfers of 0 values must be treated as normal transfers and fire the Transfer event. However, the contract implements a conditional check that prohibits transfers of 0 values. This discrepancy between the contract's implementation and the ERC20 standard may lead to inconsistencies and incompatibilities with other contracts.

```
function transfer(address _to, uint256 _value) public override
returns (bool success) {
    require(balances[msg.sender] >= _value && _value > 0,
"Insufficient balance");
        //...
}

function transferFrom(address _from, address _to, uint256
_value) public override returns (bool success) {
    require(balances[_from] >= _value &&
allowed[_from][msg.sender] >= _value && _value > 0, "Transfer
not allowed");
    //...
}
```

Recommendation

The incorrect implementation of the ERC20 standard could potentially lead to problems when interacting with the contract, as other contracts or applications that expect the ERC20 interface may not behave as expected. The team is advised to review and revise the implementation of the transfer mechanism to ensure full compliance with the ERC20 standard. https://eips.ethereum.org/EIPS/eip-20.

RBTS - Resetting Balance and Total Supply

Criticality	Minor / Informative
Location	HashnodeTestCoin.sol#L65,66
Status	Unresolved

Description

In the `HashnodeTestCoin.constructor`, the parameter `initialSupply` is added as input in the `StandardToken.constructor`. This sets the `_totalSupply` and balance of `msg.sender` to that value. However, after that, the `_totalSupply` and the balance of the sender are reassigned to fixed values.

```
constructor(uint256 initialSupply) {
        _totalSupply = initialSupply;
        balances[msg.sender] = _totalSupply;
}
//...
constructor(uint256 initialSupply) StandardToken(initialSupply)
{
        _totalSupply = 100000000000000000000000000;
        balances[msg.sender] = 100000000000000000000000000000000;
        //...
}
```

Recommendation

To maintain consistency and ensure that the intended supply is respected, it is recommended to remove or properly handle the redundant reassignment of `_totalSupply` and `balances[msg.sender]` within the `HashnodeTestCoin.constructor`.

TFPC - Token Fixed Price Concern

Criticality	Minor / Informative
Location	HashnodeTestCoin.sol#L77
Status	Unresolved

Description

The contract sells tokens at a fixed price of 10000 tokens per wei. However, the price on decentralized exchanges may fluctuate, potentially creating discrepancies (e.g. on a decentralized exchange 12000 tokens may be available for 1 wei). This could lead to opportunities for token holders or buyers that may not align with the intended business logic or the market dynamics.

```
receive() external payable {
    //...
    uint256 amount = msg.value * unitsOneEthCanBuy;
    //...
}
```

Recommendation

It is recommended that the team take into account that token prices on other decentralized applications (like a decentralized exchange) may vary from the intended price set by the contract, potentially creating discrepancies that could impact the token’s market behavior and buyer expectations.

UCM - Unnecessary Comment Messages

Criticality	Minor / Informative
Location	HashnodeTestCoin.sol#L71
Status	Unresolved

Description

The contract is using unnecessary comment messages. These comments may make it difficult to understand the source code.

```
// Adjust the price of your token here
```

Recommendation

The team is advised to carefully review the comment to improve code readability.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	HashnodeTestCoin.sol#L61
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 gas consumption of the corresponding transaction.

```
string public version = 'H1.0'
```

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	HashnodeTestCoin.sol#L21,29,38,42,48,86
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.

```
uint256 _value
address _to
address _from
address _owner
address _spender
bytes memory _extraData
```

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/stable/style-guide.html#naming-conventions.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	HashnodeTestCoin.sol#L89
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.

```
(bool successCall, ) =
_spender.call(abi.encodeWithSignature("receiveApproval(address,uint256,address,bytes)", msg.sender, _value, address(this), _extraData))
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	HashnodeTestCoin.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 ^0.8.0;
```

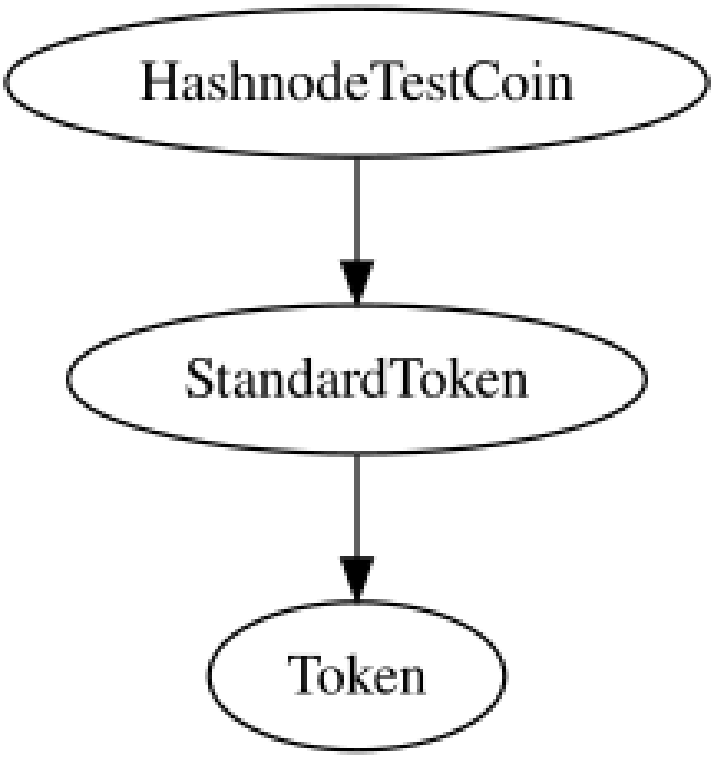
Recommendation

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.

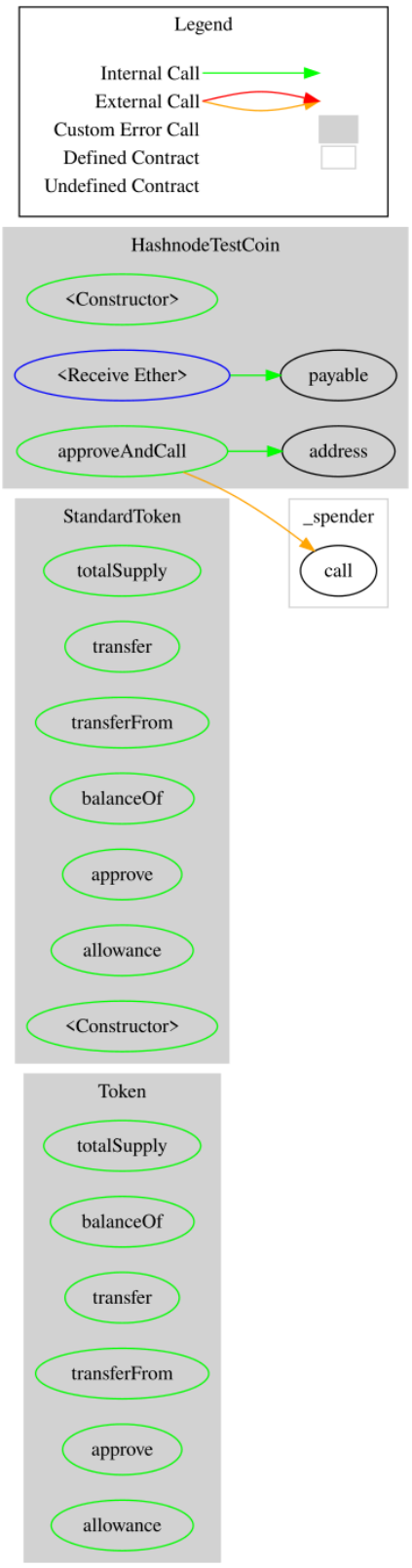
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
StandardToken	Implementation	Token		
	totalSupply	Public		-
	transfer	Public	✓	-
	transferFrom	Public	✓	-
	balanceOf	Public		-
	approve	Public	✓	-
	allowance	Public		-
		Public	✓	-
HashnodeTestCoin	Implementation	StandardToken		
		Public	✓	StandardToken
		External	Payable	-
	approveAndCall	Public	✓	-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
HashnodeTestCoin
<Constructor>
<Receive Ether>
payable
approveAndCall
address
_spender
call
StandardToken
totalSupply
transfer
transferFrom
balanceOf
approve
allowance
<Constructor>
Token
totalSupply
balanceOf
transfer
transferFrom
approve
allowance

Summary

METALBANK X contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. The Smart Contract analysis reported no compiler error but there is a critical issue with the accuracy of the total supply. The contract does not implement any 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

cyberscope.io