Cyberscope

Audit Report

METALBANK X

February 2025

Network	ETH
Address	0x1122A64Bcd9497370954Da7dA1825EE0F7185c71
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
Medium	DES	Deprecated ERC-20 Standard	Unresolved
Minor	CCR	Contract Centralization Risk	Unresolved
Minor	DFN	Deprecated Function Naming	Unresolved
Minor	MEM	Missing Error Messages	Unresolved
Minor	MTEE	Missing Transfer Event Emission	Unresolved
Minor	OSV	Outdated Solidity Version	Unresolved
Minor	TFPC	Token Fixed Price Concern	Unresolved
Minor	UCM	Unnecessary Comment Messages	Unresolved
Minor	L01	Public Function could be Declared External	Unresolved
Minor	L02	State Variables could be Declared Constant	Unresolved
Minor	L04	Conformance to Solidity Naming Conventions	Unresolved
Minor	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	v0.4.26+commit.4563c3fc
Optimization	200 runs
Explorer	https://etherscan.io/address/0x1122a64bcd9497370954da7da1825ee0f7185c71
Address	0x1122a64bcd9497370954da7da1825ee0f7185c71
Network	ETH
Symbol	MBXAU
Decimals	18
Total Supply	1.000.000.000.000
Badge Eligibility	Must Fix Medium

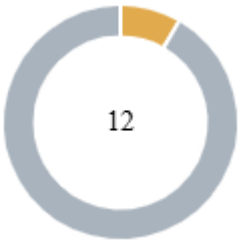
Audit Updates

Initial Audit	13 Feb 2025

Source Files

Filename	SHA256
HashnodeTestCoin.sol	cd37e0f93802c5ccd61ff04a989c2a4a659e76a74727e9800ea82d6df41fbf65

Findings Breakdown



Critical	0
Medium	1
Minor / Informative	11

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

DES - Deprecated ERC-20 Standard

Criticality	Medium
Location	HashnodeTestCoin.sol#L48,59
Status	Unresolved

Description

The contract implements an outdated version of the ERC20 Standard. Even if the contract does not explicitly state the ERC20 standards, the functionality used is very similar.

Specifically, in the newer versions of the standard, transfers of 0 values must be treated as normal transfers and fire the `Transfer` event. 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.

Additionally, the newer standards mention that transfers should throw if the balance of the spender is less than the value. In this version false is returned instead of reverting.

The case is also the same for the `transferFrom` as in the newer versions if the allowance of the spender to the caller of the function is less than the value, the function should throw.

The `totalSupply()` must also be declared as a function in the implementation of the `StandardToken` and return the `totalSupply` variable.

Additionally, recent ERC20 implementations have the following features. Transfers are not allowed to the address(0) and instead a `burn` function is used to provide similar functionality. Approvals are also not allowed to the address(0).

```
   function transfer(address _to, uint256 _value) public
returns (bool success) {
        if (balances[msg.sender] >= _value && _value > 0) {
            //...
            return true;
        } else { return false; }
    }

    function transferFrom(address _from, address _to, uint256
_value) public returns (bool success) {
        if (balances[_from] >= _value &&
allowed[_from][msg.sender] >= _value && _value > 0) {
            //...
            return true;
        } else { return false; }
    }
```

Recommendation

The team should consider using a newer version of the ERC20 standards as the latest versions introduce improvements over previous versions, offering enhanced functionalities and security features.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	HashnodeTestCoin.sol#L107,117
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 payable function of the contract, users pay an amount of ether in order to buy some 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.

```
function HashnodeTestCoin() public {
    balances[msg.sender] = 10000000000000000000000000000000;
    totalSupply = 10000000000000000000000000000000;
    //...
    fundsWallet = msg.sender;
}

function() public payable{
    //...
    require(balances[fundsWallet] >= amount);
    balances[fundsWallet] = balances[fundsWallet] - amount;
    balances[msg.sender] = 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.

DFN - Deprecated Function Naming

Criticality	Minor / Informative
Location	HashnodeTestCoin.sol#L107
Status	Unresolved

Description

The contract `HashnodeTestCoin` has a function named similar to it. In solidity's older versions these functions served the purpose of a constructor. However in the used solidity version this naming methodology is deprecated.

```
contract HashnodeTestCoin is StandardToken {
    function HashnodeTestCoin() public {/*...*/}
}
```

Recommendation

It is recommended to not use deprecated naming methodologies. This will also remove compiler warnings.

MEM - Missing Error Messages

Criticality	Minor / Informative
Location	HashnodeTestCoin.sol#L120,139
Status	Unresolved

Description

The contract is missing error messages. Specifically, there are no error messages to accurately reflect the problem, making it difficult to identify and fix the issue. As a result, the users will not be able to find the root cause of the error.

```
require(balances[fundsWallet] >= amount)
if(!_spender.call(bytes4(bytes32(keccak256("receiveApproval(address,uint256,address,bytes)"))), msg.sender, _value, this, _extraData)) { revert(); }
```

Recommendation

The team is suggested to provide a descriptive message to the errors. This message can be used to provide additional context about the error that occurred or to explain why the contract execution was halted. This can be useful for debugging and for providing more information to users that interact with the contract.

MTEE - Missing Transfer Event Emission

Criticality	Minor / Informative
Location	HashnodeTestCoin.sol#L108
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.

```
balances[msg.sender] = 100000000000000000000000000000000;
```

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.

OSV - Outdated Solidity Version

Criticality	Minor / Informative
Location	HashnodeTestCoin.sol#L1
Status	Unresolved

Description

The contract is using an outdated version of solidity. Using newer versions of Solidity is highly beneficial due to several key improvements in security, efficiency, and functionality. Newer versions come with enhanced security features that address vulnerabilities discovered in earlier versions, such as reentrancy attacks and integer overflows.

```
pragma solidity ^0.4.26;
```

Recommendation

It is recommended that the team uses the newer versions of solidity as this will provide access to better tooling, support from the community, and improved debugging capabilities, ensuring that their smart contracts are maintainable over time.

TFPC - Token Fixed Price Concern

Criticality	Minor / Informative
Location	HashnodeTestCoin.sol#L117
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.

```
function() public 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#L47,58,87,91,103,108
Status	Unresolved

Description

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

```
//If your token leaves out totalSupply and can issue more
tokens as time goes on, you need to check if it doesn't wrap.
//Replace the if with this one instead.
//if (balances[msg.sender] >= _value && balances[_to] + _value
> balances[_to]) {

//same as above. Replace this line with the following if you
want to protect against wrapping uints.
//if (balances[_from] >= _value && allowed[_from][msg.sender]
>= _value && balances[_to] + _value > balances[_to]) {

// CHANGE THIS. Update the contract name.

/*
NOTE:
The following variables are OPTIONAL vanities. One does not
have to include them.
They allow one to customise the token contract & in no way
influences the core functionality.
Some wallets/interfaces might not even bother to look at this
information.
*/
```

Recommendation

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

L01 - Public Function could be Declared External

Criticality	Minor / Informative
Location	HashnodeTestCoin.sol#L132
Status	Unresolved

Description

A public function is a function that can be called from external contracts or from within the contract itself. An external function is a function that can only be called from external contracts, and cannot be called from within the contract itself.

It's generally a good idea to declare functions as external if they are only intended to be called from external contracts, as this can help make the contract's code easier to understand and maintain. Declaring a function as external can also help to improve the contract's performance and gas consumption.

```
function approveAndCall(address _spender, uint256 _value, bytes
_extraData) public returns (bool success) {
        allowed[msg.sender][_spender] = _value;
        emit Approval(msg.sender, _spender, _value);

if(!_spender.call(bytes4(bytes32(keccak256("receiveApproval(address,uint2
56,address,bytes)"))), msg.sender, _value, this, _extraData)) { revert();
}
        return true;
    }
```

Recommendation

It's important to choose the appropriate visibility for each function based on how it is intended to be used. Declaring a function as external when it should be public, or vice versa can lead to unnecessary gas consumption.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	HashnodeTestCoin.sol#L100
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#L43,56,68,72,78,132
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 _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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	HashnodeTestCoin.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.

```
pragma solidity ^0.4.26;
```

Recommendation

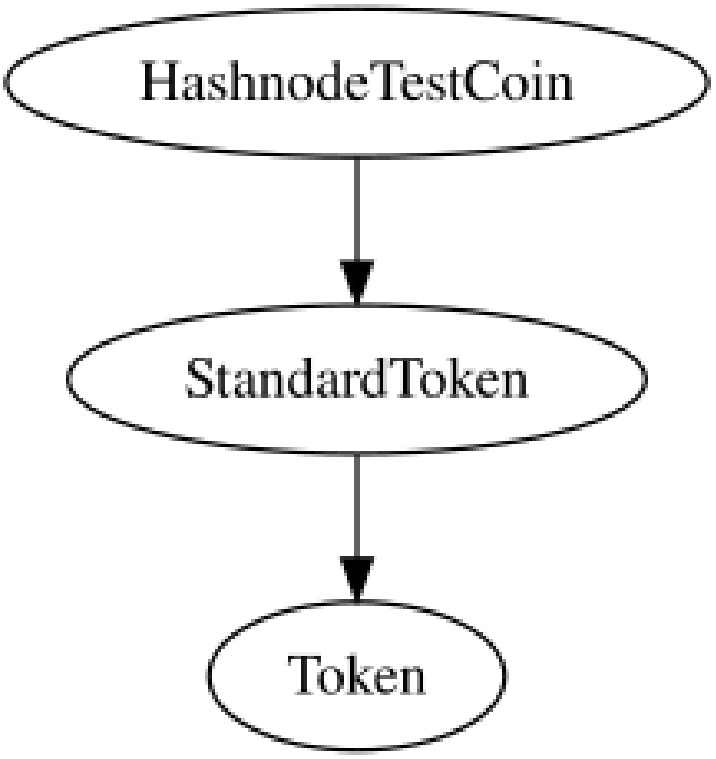
The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest 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

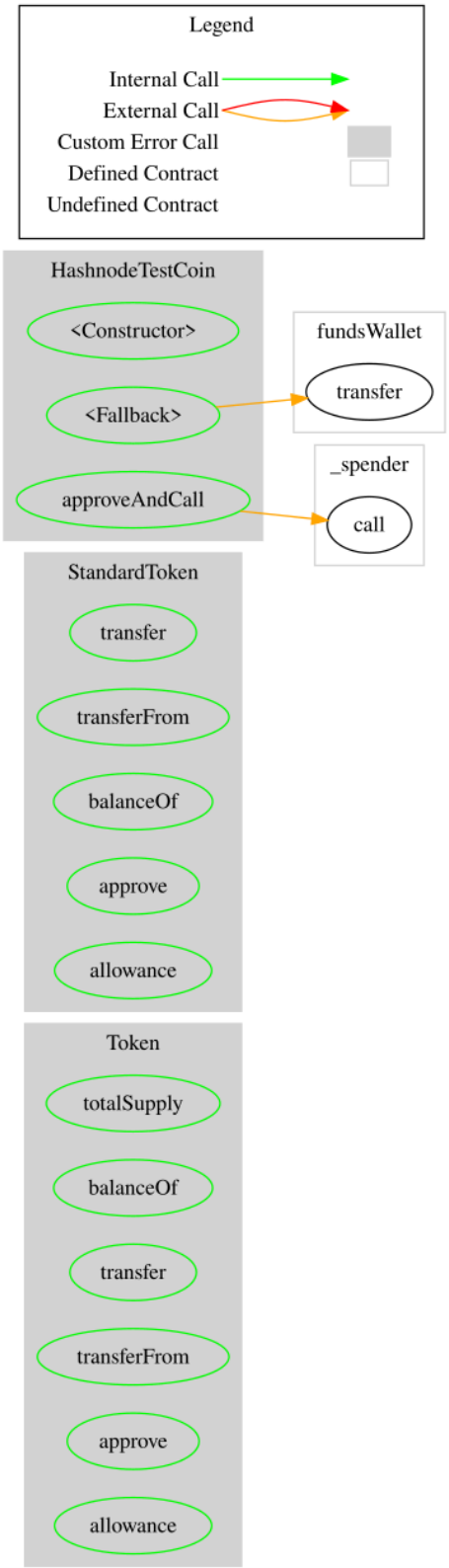
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Token	Implementation			
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	transferFrom	Public	✓	-
	approve	Public	✓	-
	allowance	Public		-
StandardToken	Implementation	Token		
	transfer	Public	✓	-
	transferFrom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

	approveAndCall	Public	✓	-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
HashnodeTestCoin
<Constructor>
<Fallback>
approveAndCall
fundsWallet
transfer
_spender
call
StandardToken
transfer
transferFrom
balanceOf
approve
allowance
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. There were some compilation warnings. 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