Cyberscope

Audit Report

Infinix Chain

January 2025

SHA256

7c99613dbb80c2a834bc40c427815f4e3e9e392b775cf74ec6fb27b3aedfc6a5

Audited by © cyberscope

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	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
●	UDO	Unnecessary Decimals Override	Unresolved
●	RCE	Redundant Code Elimination	Unresolved
●	MLI	Missing License Identifier	Unresolved
●	MCV	Missing Compiler Version	Unresolved
●	UVL	Unspecified Versions of Libraries	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	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	InfinixChain
Testing Deploy	https://testnet.bscscan.com/address/0xba1d729c5f47930cbab18795ba34308bcd35cc33
Symbol	FNX
Decimals	18
Total Supply	10.000.000.000

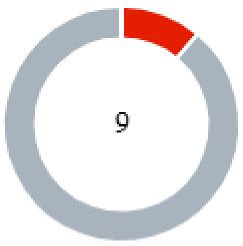
Audit Updates

Initial Audit	25 Jan 2025

Source Files

Filename	SHA256
contracts/Infinixchain_Smart_Contract_Anti_Whale_done_.sol	7c99613dbb80c2a834bc40c427815f4e3e9e392b775cf74ec6fb27b3aedfc6a5
@openzeppelin/contracts/utils/Context.sol	b2cfee351bcafd0f8f27c72d76c054df9b571b62cfac4781ed12c86354e2a56c
@openzeppelin/contracts/token/ERC20/IERC20.sol	7ebde70853ccafcf1876900dad458f46eb9444d591d39bfc58e952e2582f5587
@openzeppelin/contracts/token/ERC20/ERC20.sol	d20d52b4be98738b8aa52b5bb0f88943f62128969b33d654fbca731539a7fe0a
@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol	af5c8a77965cc82c33b7ff844deb9826166689e55dc037a7f2f790d057811990
@openzeppelin/contracts/access/Ownable.sol	a8e4e1ae19d9bd3e8b0a6d46577eec098c01fbaffd3ec1252fd20d799e73393b

Findings Breakdown



●	Critical	1
●	Medium	0
●	Minor / Informative	8

Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	1	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	8	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	Infinixchain_Smart_Contract_Anti_Whale_done_.sol#L23
Status	Unresolved

Description

The contract owner has the authority to stop the sales for all users. The owner may take advantage of it by setting the `maxWalletAmount` to zero. As a result, the contract may operate as a honeypot.

```
function setMaxWalletAmount(uint256 _maxAmount) external
onlyOwner {
    maxWalletAmount = _maxAmount;
}

// Override the _beforeTokenTransfer function to include the
anti-whale check
function _beforeTokenTransfer(address from, address to, uint256
amount) internal virtual override {
    super._beforeTokenTransfer(from, to, amount);

    // Anti-whale check: Ensure the recipient's balance doesn't
exceed the max wallet amount
    if (to != address(0) && to != owner()) {
        require(balanceOf(to) + amount <= maxWalletAmount,
"InfinixChain: Transfer amount exceeds max wallet limit");
    }
}
```

Recommendation

The contract could embody a check for not allowing setting the `maxWalletAmount` less than a reasonable amount. A suggested implementation could check that the minimum amount should be more than a fixed percentage of the total supply. 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.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

UDO - Unnecessary Decimals Override

Criticality	Minor / Informative
Status	Unresolved

Description

The contract is currently implementing an override of the decimals function, which simply returns the value 18. This override is redundant since the extending token contract already specifies 18 decimals as its standard. In the context of ERC-20 tokens, 18 decimals is a common default, and overriding this function to return the same value adds unnecessary complexity to the contract. This redundancy does not contribute to the functionality of the contract and could potentially lead to confusion about the necessity of this override.

```
function decimals() public pure override returns (uint8) {
    return 18;
}
```

Recommendation

Since the inherited ERC-20 contract already defines the decimals number, maintaining an overriding function that merely repeats this value does not contribute to the contract's effectiveness. As a result, it is recommended to remove the redundant `decimals` function from the contract. Removing this function will simplify the contract, making it more straightforward to maintain without impacting its operational capabilities.

RCE - Redundant Code Elimination

Criticality	Minor / Informative
Location	Infinixchain_Smart_Contract_Anti_Whale_done_.sol#L16,19
Status	Unresolved

Description

In the `_beforeTokenTransfer` the `super._beforeTokenTransfer` is used. However, the original `_beforeTokenTransfer` does not have any additional functionality. Therefore, the call is redundant. Similarly, in the constructor the `Ownable` contract is initiated however this is not necessary as it is inherited by the `InfinixChain` contract.

```
constructor() ERC20("InfinixChain", "FNX") Ownable() {/*...*/}
function _beforeTokenTransfer(address from, address to, uint256
amount) internal virtual override {
        super._beforeTokenTransfer(from, to, amount);
        //...
    }
```

Recommendation

Removing redundant code segments is essential for optimizing code and reducing gas costs. Optimized code also enhances future maintainability.

MLI - Missing License Identifier

Criticality	Minor / Informative
Status	Unresolved

Description

The audited smart contract is missing an explicit SPDX license identifier, which is a crucial component for ensuring the legal clarity and compliance of the code. The license identifier specifies the terms under which the code can be used, modified, and redistributed, providing essential guidance to developers and end-users. Its absence creates ambiguity regarding the rights and obligations associated with the code, potentially leading to legal disputes or misuse.

Recommendation

The team is recommended to add an SPDX license identifier to the top of the smart contract to clearly specify the terms under which the code can be used, modified, and distributed.

MCV - Missing Compiler Version

Criticality	Minor / Informative
Status	Unresolved

Description

The audited smart contract is missing an explicit pragma version directive, which is a critical element for ensuring compatibility and stability of the code. The pragma version specifies the range of compiler versions that can safely compile the contract, helping to prevent unexpected behavior or vulnerabilities due to changes in the compiler. Its absence introduces uncertainty about the intended compilation environment, potentially leading to compatibility issues, unexpected errors, or security risks.

Recommendation

It is recommended to explicitly specify the compiler version in the pragma directive and avoid using version ranges (e.g., ^) that allow multiple compiler versions. This ensures consistency in the compilation process, minimizes the risk of unexpected behavior due to compiler updates, and enhances the security and reliability of the smart contract.

UVL - Unspecified Versions of Libraries

Criticality	Minor / Informative
Location	Infinixchain_Smart_Contract_Anti_Whale_done_.sol#L1,2
Status	Unresolved

Description

The contract does not specify the versions of openzeppelin libraries that it is using.

```
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
```

Recommendation

It is recommended that the team specifies the version of the libraries that are used in the contract. Clear versioning ensures consistency, stability, and compatibility, reducing the likelihood of vulnerabilities or unexpected issues arising from library updates.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contracts/Infinixchain_Smart_Contract_Anti_Whale_done_.sol#L5
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.

```
uint256 public initialSupply = 10_000_000_000 * 10**18
```

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	contracts/Infinixchain_Smart_Contract_Anti_Whale_done_.sol#L13
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 _maxAmount
```

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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	contracts/Infinixchain_Smart_Contract_Anti_Whale_done_.sol#L14
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.

```
maxWalletAmount = _maxAmount
```

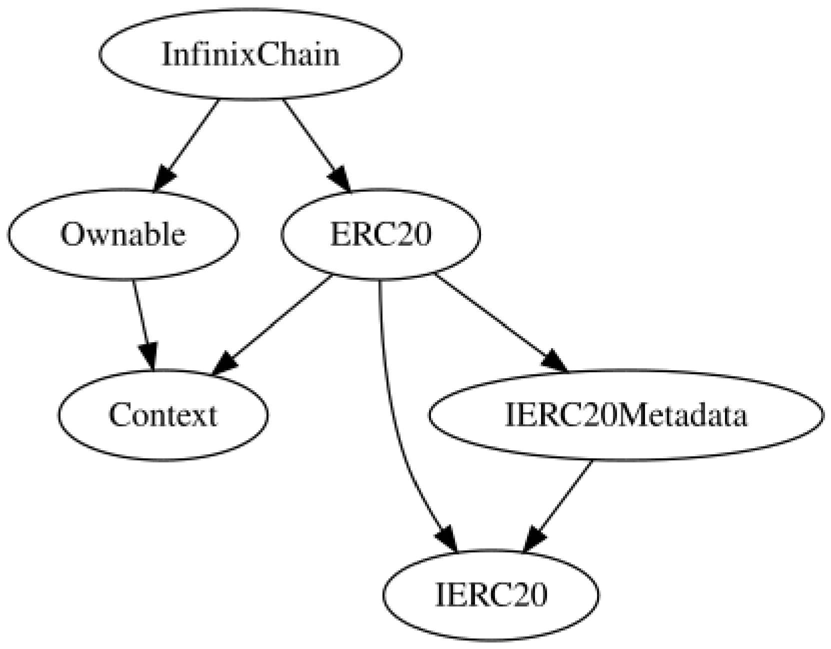
Recommendation

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.

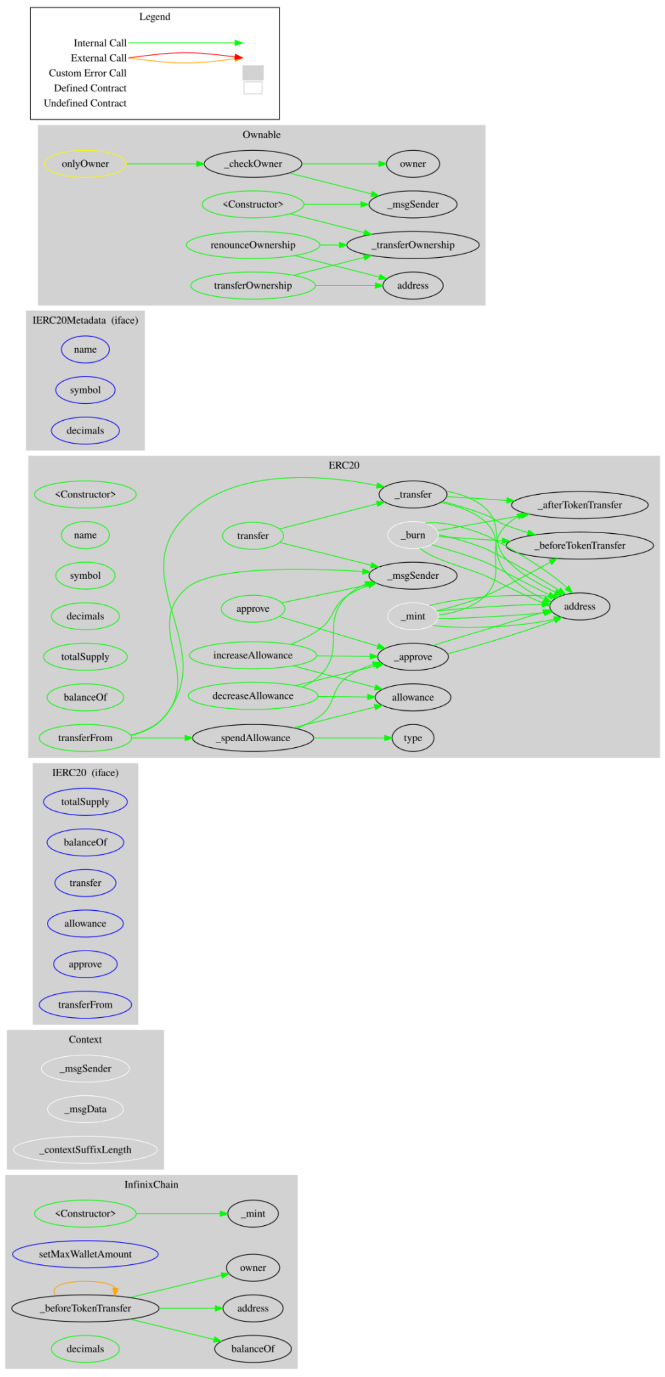
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
InfinixChain	Implementation	ERC20, Ownable		
		Public	✓	ERC20 Ownable
	setMaxWalletAmount	External	✓	onlyOwner
	_beforeTokenTransfer	Internal	✓	
	decimals	Public		-

Inheritance Graph


$$\begin{tikzcd}
 & \text{InfinixChain} \arrow[dl] \arrow[dr] & & \\
\text{Ownable} \arrow[d] & & \text{ERC20} \arrow[dll] \arrow[d] \arrow[dr] & \\
\text{Context} & & \text{IERC20} & \text{IERC20Metadata} \arrow[l]
\end{tikzcd}$$

Flow Graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
Ownable
onlyOwner
_checkOwner
owner
<Constructor>
_msgSender
renounceOwnership
_transferOwnership
transferOwnership
address
IERC20Metadata (iface)
name
symbol
decimals
ERC20
<Constructor>
_transfer
_afterTokenTransfer
name
transfer
_burn
_beforeTokenTransfer
symbol
_msgSender
decimals
approve
_mint
address
totalSupply
increaseAllowance
_approve
balanceOf
decreaseAllowance
allowance
transferFrom
_spendAllowance
type
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Context
_msgSender
_msgData
_contextSuffixLength
InfinixChain
<Constructor>
_mint
setMaxWalletAmount
owner
_beforeTokenTransfer
address
decimals
balanceOf

Summary

Infinix Chain contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. There are some functions that can be abused by the owner to stop transactions. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats. There are also zero fees in the implementation.

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

cyberscope.io