Cyberscope

A TAC Security Company

Audit Report

kaspacom lfg.kaspa

September 2025

Repository	https://github.com/fedzdev/fomoraiser
Commit	39bf50450cc69b815061233e5dd0b27da229cf0d
Audited by	© cyberscope

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

Repository	https://github.com/fedzdev/fomoraiser
Commit	39bf50450cc69b815061233e5dd0b27da229cf0d

Audit Updates

Initial Audit	12 Sep 2025

Source Files

Filename	SHA256
NewToken.sol	5a8765cc44ef97450169ff0a4874383091e5701138f24f32295bfea5f2f7dcc9
ICOFactory.sol	cc6b4c22ec8761b82da27e00beb6948a69329cbc681bc01caab8c38c14256a4b
ICOContract.sol	d7d369e817aba286c06a2850d106fe0595850ce438b1af056118150bf4a24927
libraries/VestingLibrary.sol	ccfd850b2a8f384055d5e79e9cd16ca0d6249b9fc2a3bc995ae60be5b34b90ea
libraries/ContributionLibrary.sol	d9b8b9f205294af11d44011885927704217d9f1024da17f3902b9763ce477eb2
interfaces/IWETH.sol	90162af50f99e2350663404a0bd362c47000d8cc13aefe5e30bd1fe3db9d9a7c
interfaces/IICOStructs.sol	0f4dee787b7636632d5941a76184eaec14c2cb88404e54950fc4ab120ce18aa8
interfaces/IICOContract.sol	864eafa301458b23d09ce5f1a6fd41002c4a8274792b788b5ea253067053c48f
interfaces/IFomodotbizV2Router02.sol	0c238a6ad3acb63c0a9195716525fd180d0d16ee7df8c57b56cac28a79741c72

interfaces/IFomodotbizV2Pair.sol	2c16bf86ec034edf430036d5695d33413d11695e77f5fb6c33bed945f4680fa4
interfaces/IFomodotbizV2Factory.sol	62906add4e703c6b3859fa413082a967aed8cbe204413d4f7dcd4daf9e0fa290

Overview

ICO Platform

The ICO platform consists of three core contracts: `ICOFactory`, `ICOContract`, and `NewToken`. Together, they enable the trustless deployment and management of token sale campaigns with built-in support for vesting, treasury fees, and whitelist-based access control. The platform leverages the Clones pattern from OpenZeppelin to deploy minimal proxy instances of the `ICOContract`, enabling cost-efficient scalability.

ICOContract

The `ICOContract` is responsible for managing a single ICO campaign. Once deployed by the factory, this contract handles the full lifecycle of a token sale: contributor whitelisting, accepting contributions in either native currency or ERC20 tokens, tracking allocations, finalizing the ICO, distributing tokens based on vesting schedules, refunding users on failure, and managing pre-allocations.

Each instance of `ICOContract` maintains its own configurable parameters, such as soft cap, hard cap, min/max contributions, sale start and end times, and token total supply. Token allocations are managed via a structured mechanism that supports pre-allocations with optional vesting logic for each participant.

The `claimTokens` function is used depending on the role of the caller—contributor, pre-allocated recipient, or creator—the contract routes the claim request appropriately, ensuring vesting schedules are respected and that claimed amounts are accurately tracked. The platform supports linear vesting with an optional cliff and initial unlock percentage. All token prices are dynamically calculated based on the ratio between hard cap and total tokens available for sale, with support for 18-decimal precision.

Refunds are supported for contributors in case the sale fails to meet the soft cap. The creator is then able to reclaim unsold tokens. For treasury management, the contract allocates a fixed percentage of raised funds, which is transferred equaly to two treasury addresses.

The `ICOContract` has a built-in whitelisting system. If enabled, only approved addresses may contribute. The creator can manage the whitelist dynamically or disable it permanently.

ICOFactory

The `ICOFactory` is responsible for deploying `ICOContract` clones and their corresponding `NewToken` ERC20 tokens. When a user creates a new ICO through the factory, the system first validates the provided configuration, ensuring logical correctness.

Once validated, the factory deploys a `NewToken` instance and mints the full supply to the new ICO contract. It then deploys a proxy of `ICOContract` and initializes it with the creator's parameters, including vesting settings, treasury address, and optional whitelist. Ownership of the token contract is transferred to the ICO contract to enable autonomous control over token distribution.

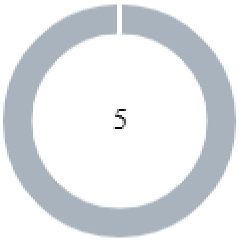
The factory charges a deployment fee (in native currency), which is immediately forwarded to the treasury address. It also includes administrative functions for updating the template contract and treasury.

Only the ICO creator may activate their ICO via the factory once the campaign start time has passed.

NewToken

The `NewToken` contract is a standard ERC20 token with minting and ownership managed by the `ICOFactory`. Each ICO gets its own unique token, with a total supply minted directly to the ICO contract. Ownership of the token is handed over to the ICO contract so that it can enforce vesting, token claims, and treasury allocations independently.

Findings Breakdown



🔴	Critical	0
🟠	Medium	0
●	Minor / Informative	5

	Severity	Unresolved	Acknowledged	Resolved	Other
🔴	Critical	0	0	0	0
🟠	Medium	0	0	0	0
●	Minor / Informative	2	3	0	0

Diagnostics

🔴 Critical 🟠 Medium ● Minor / Informative

Severity	Code	Description	Status
●	AME	Address Manipulation Exploit	Acknowledged
●	CCR	Contract Centralization Risk	Acknowledged
●	PMTT	Potential Mocked Token Template	Unresolved
●	TFFI	Treasury Funding Fee Inconsistency	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Acknowledged

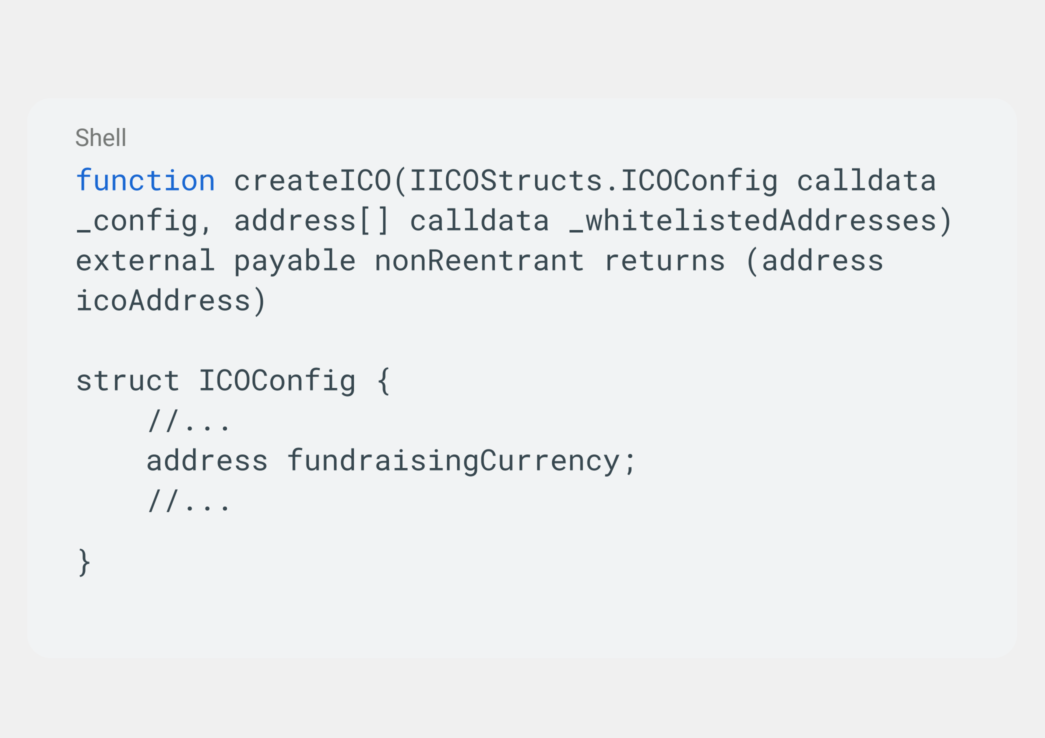
AME - Address Manipulation Exploit

Criticality	Minor / Informative
Location	ICOFactory.sol#L100
Status	Acknowledged

Description

The contract's design includes functions that accept external contract addresses as parameters without performing adequate validation or authenticity checks. This lack of verification introduces a significant security risk, as input addresses could be controlled by attackers and point to malicious contracts. Such vulnerabilities could enable attackers to exploit these functions, potentially leading to unauthorized actions or the execution of malicious code under the guise of legitimate operations.

Specifically the creator could use a malicious token as `fundraisingCurrency` to harm the contracts operations. This could happen during the user's contribution, refund, ICO's finalization or during the claim of preallocated funds.



Shell

```
function createICO(IICOStructs.ICOConfig calldata
_config, address[] calldata _whitelistedAddresses)
external payable nonReentrant returns (address
icoAddress)

struct ICOConfig {
    //...
    address fundraisingCurrency;
    //...
}
```

Recommendation

To mitigate this risk and enhance the contract's security posture, it is imperative to incorporate comprehensive validation mechanisms for any external contract addresses passed as parameters to functions. This could include checks against a whitelist of approved addresses, verification that the address implements a specific contract interface or other methods that confirm the legitimacy and integrity of the external contract. Implementing such validations helps prevent malicious exploits and ensures that only trusted contracts can interact with sensitive functions.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	ICOFactory.sol#L178,203,209,217,223,235,248 ICOContract.sol#L322,392,453,466,477,486,496,528,941,944
Status	Acknowledged

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.

Shell

```
function activateICO(address _icoAddress) external
nonReentrant {
    //...
    if (msg.sender != info.creator) { revert
ICOFactory__Unauthorized(); }
    //...
}
function setTemplate(address _newTemplate)
external onlyOwner
function setFee(uint256 _newFee) external
onlyOwner
function setTreasuryAddress(address _newTreasury)
external onlyOwner
function setPartnerTreasuryAddress(address
_newTreasury) external onlyOwner
function setTreasuryFundingFee(uint16 _newFeeBps)
external onlyOwner

function renounceOwnershipPermanently() external
onlyOwner
```

Shell

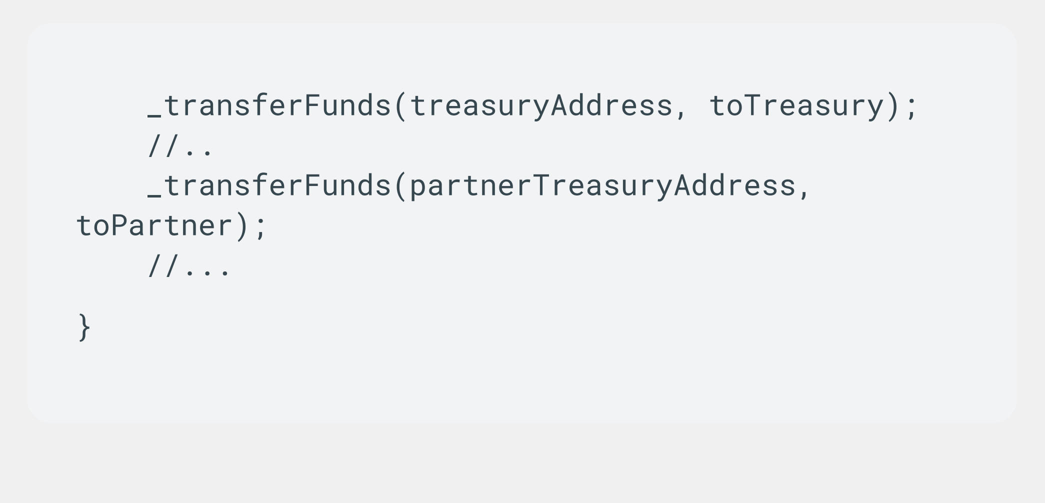
```
function activate() external nonReentrant
onlyFactory
function reclaimTokensOnFailure() external
nonReentrant onlyCreator checkState(State.Failed)
function addToWhitelist(address _account) external
onlyCreator whenWhitelistActive
function removeFromWhitelist(address _account)
external onlyCreator whenWhitelistActive
function deactivateWhitelist() external
onlyCreator whenWhitelistActive
function activateWhitelist() external onlyCreator
whenWhitelistNotActive
function addManyToWhitelist(address[] calldata
_accounts) external onlyCreator
whenWhitelistActive

function removeManyFromWhitelist(address[]
calldata _accounts) external onlyCreator
whenWhitelistActive
```

Additionally, in the `ICOContract` the treasury addresses can be contract addresses that revert during the receive of native funds resulting in the contract not being able to finalize.

Shell

```
function _handleTreasuryFees() internal {
    //...
```



```
    _transferFunds(treasuryAddress, toTreasury);
    //..
    _transferFunds(partnerTreasuryAddress,
toPartner);
    //...
}
```

Recommendation

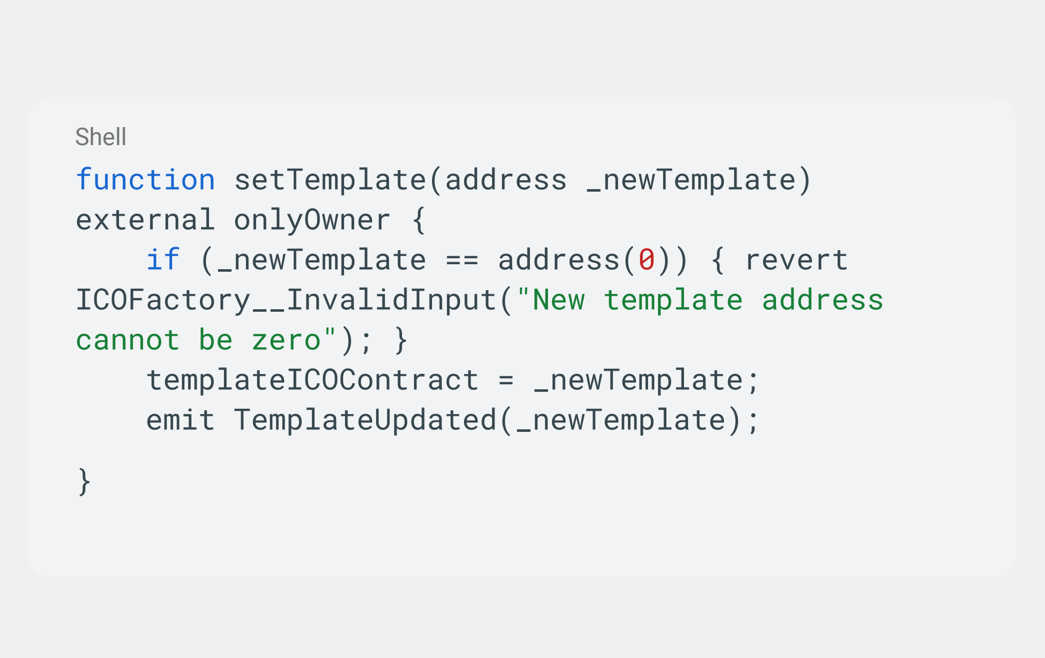
To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential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

PMTT - Potential Mocked Token Template

Criticality	Minor / Informative
Location	ICOFactory.sol#L203
Status	Unresolved

Description

The contract includes a method that allows the owner to modify the template address of the token. While this feature provides flexibility, it introduces a security threat. The owner could set the template address to any contract that implements the token's interface, potentially containing malicious code. In the event of a transaction triggering the token's functionality the transaction may be manipulated.



Shell

```
function setTemplate(address _newTemplate)
external onlyOwner {
    if (_newTemplate == address(0)) { revert
ICOFactory__InvalidInput("New template address
cannot be zero"); }
    templateICOContract = _newTemplate;
    emit TemplateUpdated(_newTemplate);

}
```

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.

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:

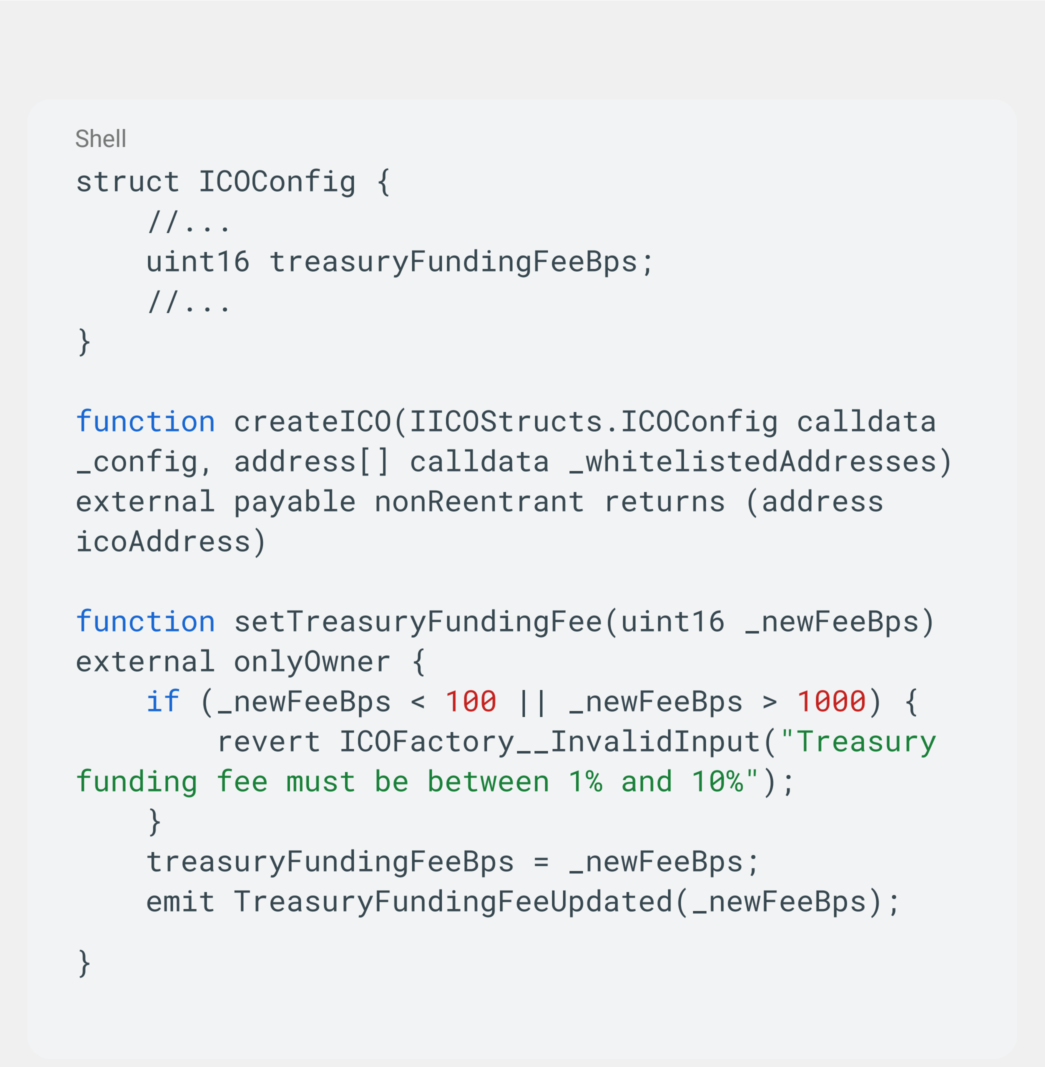
- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

TFFI - Treasury Funding Fee Inconsistency

Criticality	Minor / Informative
Location	ICOFactory.sol#L100,235
Status	Unresolved

Description

The `ICOFactory` contract has a function named `setTreasuryFundingFee`. This function allows the owner to set `treasuryFundingFeeBps`. However in the configuration of the ICO the caller of `createICO` is able to add as input a different `treasuryFundingFeeBps`.



Shell

```
struct ICOConfig {
    //...
    uint16 treasuryFundingFeeBps;
    //...
}

function createICO(IICOStructs.ICOConfig calldata
_config, address[] calldata _whitelistedAddresses)
external payable nonReentrant returns (address
icoAddress)

function setTreasuryFundingFee(uint16 _newFeeBps)
external onlyOwner {
    if (_newFeeBps < 100 || _newFeeBps > 1000) {
        revert ICOFactory__InvalidInput("Treasury
funding fee must be between 1% and 10%");
    }
    treasuryFundingFeeBps = _newFeeBps;
    emit TreasuryFundingFeeUpdated(_newFeeBps);

}
```

Recommendation

The teams should choose between allowing users to define a `treasuryFundingFeeBps` or to be set by the owner with the `setTreasuryFundingFee` .

L04 - Conformance to Solidity Naming Conventions

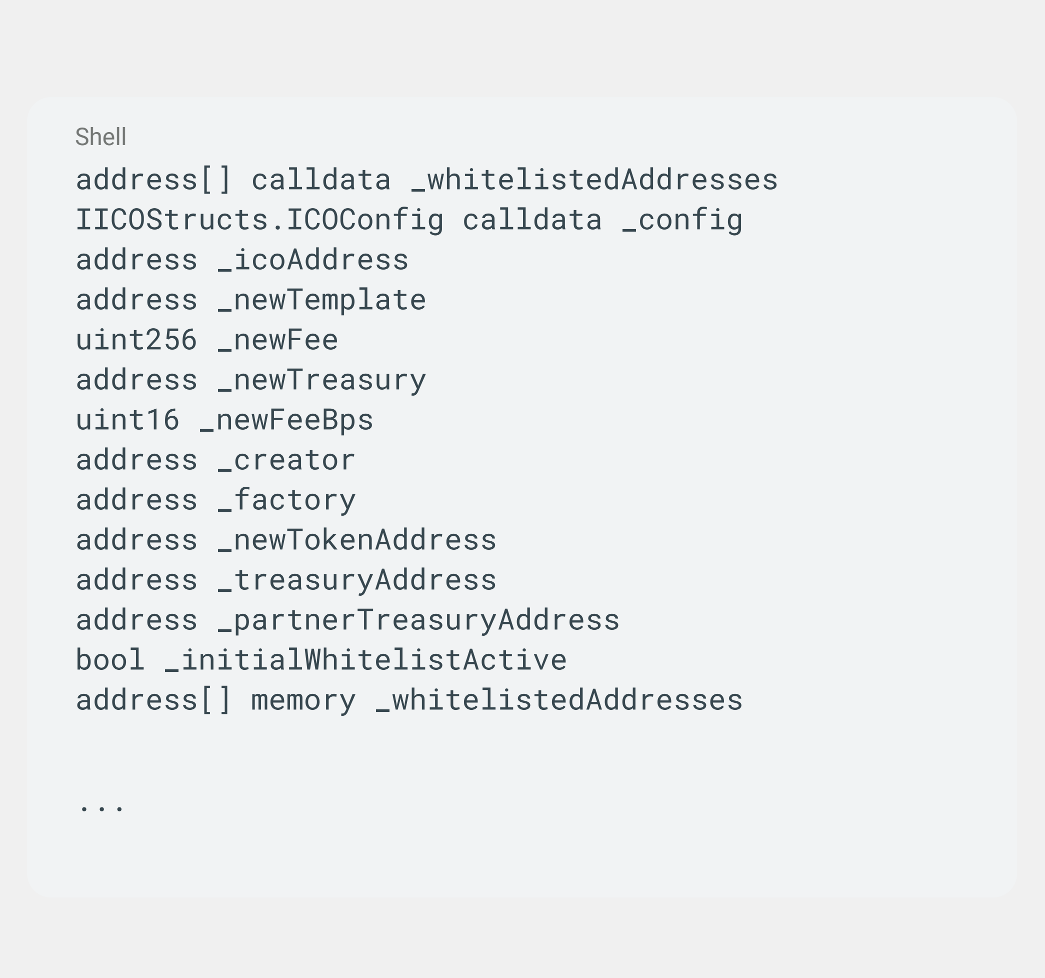
Criticality	Minor / Informative
Location	ICOFactory.sol#L100,178,203,209,217,223,235,253 ICOContract.sol#L180,181,182,183,184,185,186,187,278,453,466,496,528,571,634
Status	Acknowledged

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.



Shell

```
address[] calldata _whitelistedAddresses
IICOStructs.ICOConfig calldata _config
address _icoAddress
address _newTemplate
uint256 _newFee
address _newTreasury
uint16 _newFeeBps
address _creator
address _factory
address _newTokenAddress
address _treasuryAddress
address _partnerTreasuryAddress
bool _initialWhitelistActive
address[] memory _whitelistedAddresses


...
```

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.

Functions Analysis

Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
NewToken	Implementation	ERC20, Ownable		
		Public	✓	ERC20 Ownable
	mint	Public	✓	onlyOwner
ICOFactory	Implementation	Ownable, ReentrancyGuard		
		Public	✓	Ownable
	createICO	External	Payable	nonReentrant
	activateICO	External	✓	nonReentrant
	setTemplate	External	✓	onlyOwner
	setFee	External	✓	onlyOwner
	setTreasuryAddress	External	✓	onlyOwner
	setPartnerTreasuryAddress	External	✓	onlyOwner
	setTreasuryFundingFee	External	✓	onlyOwner
	renounceOwnershipPermanently	External	✓	onlyOwner
	getICOInfo	External		-
	_validateICOConfig	Internal		
	_validateFundraisingCurrency	Internal		
	_isValidERC20Token	Internal		
	_validateTokenCharacteristics	Internal		

ICOContract	Implementation	Initializable, ReentrancyGuard		
		Public	✓	-
	initialize	External	✓	initializer
	contribute	External	Payable	nonReentrant
	activate	External	✓	nonReentrant onlyFactory
	finalize	External	✓	nonReentrant checkState
	refund	External	✓	nonReentrant checkState
	reclaimTokensOnFailure	External	✓	nonReentrant onlyCreator checkState
	claimPreAllocatedFunding	External	✓	nonReentrant checkState
	addToWhitelist	External	✓	onlyCreator whenWhitelistActive
	removeFromWhitelist	External	✓	onlyCreator whenWhitelistActive
	deactivateWhitelist	External	✓	onlyCreator whenWhitelistActive
	activateWhitelist	External	✓	onlyCreator whenWhitelistNotActive
	addManyToWhitelist	External	✓	onlyCreator whenWhitelistActive
	removeManyFromWhitelist	External	✓	onlyCreator whenWhitelistActive
	claimTokens	External	✓	nonReentrant checkState
	_processContribution	Internal	✓	

	getTokenSoldAddress	External		-
	getTotalTokensForSale	External		-
	getState	External		-
	getStartTime	External		-
	getRemainingTokens	External		-
	getClaimableAmount	External		-
	getVestingStatus	External		-
	_getVestedAmount	Internal		
	_validateAndProcessTokenAllocations	Internal	✓	
	_validateAndProcessFundingAllocations	Internal	✓	
	_processTokenClaim	Internal	✓	
	_transferFunds	Internal	✓	
	_handleTreasuryFees	Internal	✓	
	_isValidClaimType	Internal		
	_processVestedClaimWithTracking	Internal	✓	
VestingLibrary	Library			
	getVestedAmount	External		-
	calculateClaimableAmount	External		-
ContributionLibrary	Library			
	processContribution	External	✓	-
	_sendRefund	Internal	✓	

IICOStructs	Interface			
IICOContract	Interface			
	initialize	External	✓	-
	activate	External	✓	-
	getState	External		-
	getStartTime	External		-
	getTokenSoldAddress	External		-
	getTotalTokensForSale	External		-
	whitelistActive	External		-
	isWhitelisted	External		-
	addToWhitelist	External	✓	-
	removeFromWhitelist	External	✓	-
	removeManyFromWhitelist	External	✓	-
	deactivateWhitelist	External	✓	-
	activateWhitelist	External	✓	-
	addManyToWhitelist	External	✓	-
	contribute	External	Payable	-
	contributeWithToken	External	✓	-
	claimTokens	External	✓	-
	claimPreAllocatedFunding	External	✓	-
	finalize	External	✓	-
	refund	External	✓	-
	reclaimTokensOnFailure	External	✓	-
	creator	External		-

	fundraisingCurrency	External		-
	treasuryAddress	External		-
	tokenSold	External		-
	endTime	External		-
	tokensAvailableForSale	External		-
	softCap	External		-
	hardCap	External		-
	minContribution	External		-
	maxContribution	External		-
	tokenTotalSupply	External		-
	totalRaised	External		-
	treasuryFundingFeeBps	External		-
	contributions	External		-
	tokenAllocations	External		-
	getClaimableAmount	External		-
	getVestingStatus	External		-
	shortDescription	External		-
	fullDescription	External		-
	twitter	External		-
	telegram	External		-
	website	External		-
	discord	External		-
	docs	External		-
	logoCid	External		-
	bannerCid	External		-

Inheritance Graph

The inheritance graph of kaspacom lfg.kaspa can be found here:

kaspacom lfg.kaspa_inheritance_graph.png

Flow Graph

The flow graph of kaspacom lfg.kaspa can be found here:

kaspacom lfg.kaspa_flow_graph.png

Summary

kaspacom lfg.kaspa contract implements a token and ICO creation mechanism. This audit investigates security issues, business logic concerns and potential improvements.

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 TAC 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.

A ***TAC Security*** Company

The Cyberscope team

cyberscope.io