Cyberscope

A TAC Security Company

Audit Report

Sqwad

October 2025

Network	ETH
Address	0xdE16fcd228a9d73fFc3c96e5b7fF4329e9dE7e7c
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

Explorer	https://etherscan.io/address/0xde16fcd228a9d73ffc3c96e5b7ff4329e9de7e7c

Audit Updates

Initial Audit	09 Oct 2025

Source Files

Filename	SHA256
TokenICO.sol	2e48244e70d4e74113b428653e03864a93086c20974e553bf7de6b285706cc66

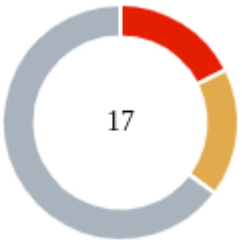
Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Critical	MBS	Missing Balance Segregation	Acknowledged
Critical	IPD	Indefinite Presale Duration	Acknowledged
Critical	MSP	Mutable Stake Parameters	Acknowledged
Medium	MSP	Mutable Sale Parameters	Acknowledged
Medium	TSI	Tokens Sufficiency Insurance	Acknowledged
Medium	OCTD	Transfers Contract's Tokens	Acknowledged
Minor / Informative	CR	Code Repetition	Acknowledged
Minor / Informative	CCR	Contract Centralization Risk	Acknowledged
Minor / Informative	MC	Missing Check	Acknowledged
Minor / Informative	OVA	Overlapping Vesting Allocations	Acknowledged
Minor / Informative	PF	Pausable Functionality	Acknowledged
Minor / Informative	PTAI	Potential Transfer Amount Inconsistency	Acknowledged
Minor / Informative	TDM	Token Decimals Mismatch	Acknowledged
Minor / Informative	L04	Conformance to Solidity Naming Conventions	Acknowledged
Minor / Informative	L06	Missing Events Access Control	Acknowledged
Minor / Informative	L07	Missing Events Arithmetic	Acknowledged

●	L19	Stable Compiler Version	Acknowledged

Findings Breakdown



Critical	3
Medium	3
Minor / Informative	11

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	3	0	0
Medium	0	3	0	0
Minor / Informative	0	11	0	0

MBS - Missing Balance Segregation

Criticality	Critical
Location	TokenICO.sol#L237,313
Status	Acknowledged

Description

The contract manages token balances for multiple purposes, including token sales, reward distributions, and staked balances. However, it does not differentiate between these types of balances. As a result, it is possible for users to purchase tokens that have already been staked by other users. This inconsistency undermines the presale logic and could potentially lead to financial losses for users.

```
Shell
function buyTokens() external payable
whenNotPaused nonReentrant {
...

}
```

```
Shell
function stakeTokens(uint256 amount, uint256 lockDuration)
external whenNotPaused nonReentrant {...}
```

Recommendation

The team is advised to differentiate between the underlying balances within the contract to ensure consistency and maintain user trust. It is recommended to implement restrictions that prevent staking before the presale is finalized, while also ensuring that no tokens can be purchased once staking has commenced.

Team Update

The team has acknowledged that this is not a security issue and states:

We acknowledge this finding. To mitigate this, presale stages are manually controlled and finalized by the team. Once the final stage concludes, the presale contract will be explicitly marked as closed, and all token sale functions will be disabled to prevent further purchases. This ensures that no tokens — including staked tokens — can be bought again at earlier stage prices after the sale has ended.

IPD - Indefinite Presale Duration

Criticality	Critical
Location	TokenICO.sol#L216
Status	Acknowledged

Description

The contract implements the `getCurrentStage` function to retrieve the current stage of the presale. The latter ensures that if all stages have been finalized and the `currentStageIndex` is set to a valid index the presale can be executed for that index.

However, the `currentStageIndex` variable is initialized to zero by default. As a result, when all stages have been finalized, the contract may reset to the initial stage, potentially allowing the presale to continue at the lowest price. Additionally, the owner cannot advance the stage beyond the valid range, causing the presale to remain active indefinitely for the selected `currentStageIndex`.

```shell
function getCurrentStage() public view returns (uint256 index, Stage memory s, bool active) {
for (uint256 i = 0; i < stages.length; ++i) {
if (block.timestamp >= stages[i].start && block.timestamp <= stages[i].end) {
return (i, stages[i], true);
}
}
if (currentStageIndex < stages.length) {
return (currentStageIndex, stages[currentStageIndex], false);
}
return (type(uint256).max, Stage("", 0, 0, 0, false), false);}
```

Recommendation

The team is advised to revise the implementation to ensure that the presale mechanism can be properly finalized. This can be achieved by preventing the execution of any past presale stages.

Team Update

The team has acknowledged that this is not a security issue and states:

We acknowledge this recommendation and have implemented a controlled approach to manage presale stages manually. Each public presale phase will be properly finalized before the next begins. Automating this process could introduce potential edge cases or unintended execution of outdated stages. By handling this manually, we ensure greater operational control, accuracy, and flexibility in finalizing each presale stage safely.

MSP - Mutable Stake Parameters

Criticality	Critical
Location	TokenICO.sol#L142,313,327
Status	Acknowledged

Description

The contract implements a staking mechanism that allows the owner to modify key parameters, including the token being staked. While this level of centralization provides flexibility, it also introduces risks of uncertainty and inconsistency in the code. If these stake characteristics are altered once staking is active, user allocations and funds could be affected, potentially undermining overall user trust in the system.

```shell
function setSaleToken(address _token) external onlyOwner
{...}
function stakeTokens(uint256 amount, uint256 lockDuration)
external whenNotPaused nonReentrant {...}

function unstake(uint256 stakeIndex) external whenNotPaused
nonReentrant {...}
```

Recommendation

The team is advised to restrict such modifications to the period before staking is activated. Altering staking characteristics while the process is active could adversely affect user funds and compromise the integrity of the staking mechanism.

Team Update

The team has acknowledged that this is not a security issue and states:

We agree with this recommendation. All staking parameters will be finalized before staking activation. Once staking is live, no modifications to these parameters will be permitted. This ensures that user funds remain protected and that the staking process maintains full integrity and transparency throughout its lifecycle.

MSP - Mutable Sale Parameters

Criticality	Medium
Location	TokenICO.sol#L142,156,194
Status	Acknowledged

Description

The contract implements a presale mechanism that allows the owner to modify key parameters, including the token being sold, its decimal precision, and its price. While this level of centralization provides flexibility, it also introduces risks of uncertainty and inconsistency in the code. If these presale characteristics are altered once the sale is active, user allocations and funds could be affected, potentially undermining overall user trust in the system.

```shell
function setSaleToken(address _token) external
onlyOwner {...}
function setSaleTokenDecimals(uint8 d) external
onlyOwner {...}
function updateStage(
uint256 index,
string calldata name,
uint256 start,
uint256 end,
uint256 priceWeiPerTokenUnit,
bool whitelistOnly

) external onlyOwner {...}
```

Recommendation

The team is advised to restrict such modifications to the period before the sale is activated. Altering presale characteristics while the sale is active could adversely affect user funds and compromise the integrity of the presale process.

TSI - Tokens Sufficiency Insurance

Criticality	Medium
Location	TokenICO.sol#L264,299,340
Status	Acknowledged

Description

The tokens are not held within the contract itself. Instead, the contract is designed to provide the tokens from an external administrator. While external administration can provide flexibility, it introduces a dependency on the administrator's actions, which can lead to various issues and centralization risks. Specifically, if the necessary tokens to cover bought amounts, staked assets and claimed rewards are not in the system, the contract may not be able to operate.

```shell
bool ok = tokenContract.transfer(msg.sender,
tokenAmount);
```

```shell
IERC20 tokenContract = IERC20(saleToken);

bool ok = tokenContract.transfer(beneficiary,
releasable);
```

```shell
require(contractBal >= principal + reward,
"insufficient reward balance");
```

Recommendation

It is recommended to consider implementing a more decentralized and automated approach for handling the contract tokens. One possible solution is to hold the tokens within the contract itself. If the contract guarantees the process it can enhance its reliability, security, and participant trust, ultimately leading to a more successful and efficient process.

Team Update

The team has acknowledged that this is not a security issue and states:

We acknowledge the recommendation and have opted for a controlled, manual token allocation strategy to balance flexibility with security. Tokens will be manually transferred to the presale contract by the administrator before each presale phase to ensure that sufficient liquidity is available for all purchases, staking, and reward distributions. This approach reduces dependency on continuous external administration while maintaining clear oversight and operational safety.

OCTD - Transfers Contract's Tokens

Criticality	Medium
Location	TokenICO.sol#L274,365
Status	Acknowledged

Description

Through the `createVesting` method the owner has authority to access the tokens in the contract. Specifically the owner is able to claim all tokens in the contract by allocating a vesting schedule to an owner control address. In addition, the owner can modify the `saleToken` and via the `rescueTokens` withdraw the tokens in the contract.

```
Shell
function createVesting(address beneficiary, uint256 amount,
uint256 start, uint256 duration) external onlyOwner {
...
}
```

```
Shell
function rescueTokens(address tokenAddress) external
onlyOwner nonReentrant {...}
```

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.

CR - Code Repetition

Criticality	Minor / Informative
Location	TokenICO.sol#L342,344
Status	Acknowledged

Description

The contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

```
Shell
bool ok1 = tokenContract.transfer(msg.sender,
principal);
if (!ok1) revert TokenTransferFailed();
bool ok2 = tokenContract.transfer(msg.sender,
reward);

if (!ok2) revert TokenTransferFailed();
```

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help reduce the complexity and size of the contract. For instance, the contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	TokenICO.sol#L142,156,160,165,170,175,181,194,209,229,274,350,357,365
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 setSaleToken(address _token) external onlyOwner
{...}
function setSaleTokenDecimals(uint8 d) external onlyOwner
{...}
function setTreasury(address _treasury) external onlyOwner
{...}
function setLiquidityWallet(address _liquidity) external
onlyOwner {...}
function pause() external onlyOwner {...}
function unpause() external onlyOwner {...}
function addStage( string calldata name,
uint256 start,
uint256 end,
uint256 priceWeiPerTokenUnit,
bool whitelistOnly
) external onlyOwner {...}
function updateStage(
uint256 index,
string calldata name,
uint256 start,
uint256 end,
uint256 priceWeiPerTokenUnit,
bool whitelistOnly ) external onlyOwner {...}
function advanceToStage(uint256 index) external onlyOwner
{...}
function setWhitelist(address[] calldata accounts, bool
allowed) external onlyOwner {...}
function createVesting(address beneficiary, uint256 amount,
uint256 start, uint256 duration) external onlyOwner {...}
function transferOwnership(address newOwner) external
onlyOwner {...}
function transferOwnership(address newOwner) external
onlyOwner {...}
function withdrawETH(uint256 amount, address to) external
onlyOwner nonReentrant {...}

function rescueTokens(address tokenAddress) external

onlyOwner nonReentrant {...}
```

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.

MC - Missing Check

Criticality	Minor / Informative
Location	TokenICO.sol#L181,194,229,274
Status	Acknowledged

Description

The contract processes variables that have not been properly validated or checked for correct structure, potentially introducing vulnerabilities. Specifically, it does not ensure that vesting schedules start after the current timestamp and end after their start date, nor does it prevent presale stages from overlapping, and it does not consistently validate against the zero address. Such inconsistencies may lead to errors in contract execution, affecting overall performance and undermining user trust.

Shell

```
function createVesting(address beneficiary, uint256 amount,
uint256 start, uint256 duration) external onlyOwner {
...
}

function updateStage(
uint256 index,
string calldata name,
uint256 start,
uint256 end,
uint256 priceWeiPerTokenUnit,
bool whitelistOnly
) external onlyOwner {
...
}

function addStage(
string calldata name,
uint256 start,
uint256 end,
uint256 priceWeiPerTokenUnit,
bool whitelistOnly
) external onlyOwner {
...
}

function setWhitelist(address[] calldata accounts, bool
allowed) external onlyOwner {
for (uint256 i = 0; i < accounts.length; ++i) {
whitelist[accounts[i]] = allowed;
emit WhitelistSet(accounts[i], allowed);
}

}
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

OVA - Overlapping Vesting Allocations

Criticality	Minor / Informative
Location	TokenICO.sol#L274
Status	Acknowledged

Description

The contract implements the `createVesting` method, which allows the owner to allocate a vested amount to a beneficiary. If the beneficiary already has an existing vesting allocation, the method increases the total allocation without updating the vesting duration. As a result, a portion,or potentially all of the newly allocated tokens, may become immediately available.

```shell
function createVesting(address beneficiary, uint256 amount,
uint256 start, uint256 duration) external onlyOwner {
...
if (v.totalAmount == 0) {
...
} else {
v.totalAmount += amount;}
...
}
```

Recommendation

The team is advised to review the current vesting allocation logic. If the intention is for the allocated amount to remain vested throughout the entire vesting period, the implementation should ensure this behavior. Doing so will maintain consistency in the token release schedule and prevent unintended early access to vested tokens.

PF - Pausable Functionality

Criticality	Minor / Informative
Location	TokenICO.sol#L170,175
Status	Acknowledged

Description

The contract owner has the authority to pause critical functionalities, including staking and unstaking operations.

```shell
function pause() external onlyOwner {
paused = true;
emit Paused(msg.sender);
}

function unpause() external onlyOwner {
paused = false;
emit Unpaused(msg.sender);

}
```

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.

PTAI - Potential Transfer Amount Inconsistency

Criticality	Minor / Informative
Location	TokenICO.sol#L320,323
Status	Acknowledged

Description

The `transfer()` and `transferFrom()` functions are used to transfer a specified amount of tokens to an address. The fee or tax is an amount that is charged to the sender of an ERC20 token when tokens are transferred to another address. According to the specification, the transferred amount could potentially be less than the expected amount. This may produce inconsistency between the expected and the actual behavior.

The following example depicts the diversion between the expected and actual amount.

Tax	Amount	Expected	Actual
No Tax	100	100	100
10% Tax	100	100	90

```shell
function stakeTokens(uint256 amount, uint256 lockDuration)
external whenNotPaused nonReentrant {
...
bool ok = tokenContract.transferFrom(msg.sender,
address(this), amount);
if (!ok) revert TokenTransferFailed();
stakes[msg.sender].push(StakeInfo(amount, block.timestamp,
lockDuration, true));
emit Staked(msg.sender, amount, lockDuration);

}
```

Recommendation

The team is advised to take into consideration the actual amount that has been transferred instead of the expected.

It is important to note that an ERC20 transfer tax is not a standard feature of the ERC20 specification, and it is not universally implemented by all ERC20 contracts. Therefore, the contract could produce the actual amount by calculating the difference between the transfer call.

`Actual Transferred Amount = Balance After Transfer - Balance Before Transfer`

TDM - Token Decimals Mismatch

Criticality	Minor / Informative
Location	TokenICO.sol#L156
Status	Acknowledged

Description

The contract owner can update the decimal precision used in presale calculations. This variable can be modified unilaterally, without adjusting the sale price accordingly. If such a change occurs, the contract may transfer amounts that do not reflect the intended design.

```shell
function setSaleTokenDecimals(uint8 d) external
onlyOwner {
saleTokenDecimals = d;

}
```

Recommendation

The team is advised to ensure price consistency at all times. This can be achieved by synchronously updating both the decimal precision and the underlying token price, thereby maintaining stable and predictable pricing within the contract.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	TokenICO.sol#L24,142,160,165
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
contract TokenICO_v2
...
address _token
...
address _treasury
...
address _liquidity
...
```

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.

L06 - Missing Events Access Control

Criticality	Minor / Informative
Location	TokenICO.sol#L352
Status	Acknowledged

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. There are functions that have no event emitted, so it is difficult to track off-chain changes.

```
Shell
owner = newOwner
```

Recommendation

To avoid this issue, 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.

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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	TokenICO.sol#L157
Status	Acknowledged

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.

```
Shell
saleTokenDecimals = d
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	TokenICO.sol#L2
Status	Acknowledged

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.

```
Shell
pragma solidity ^0.8.19;
```

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.

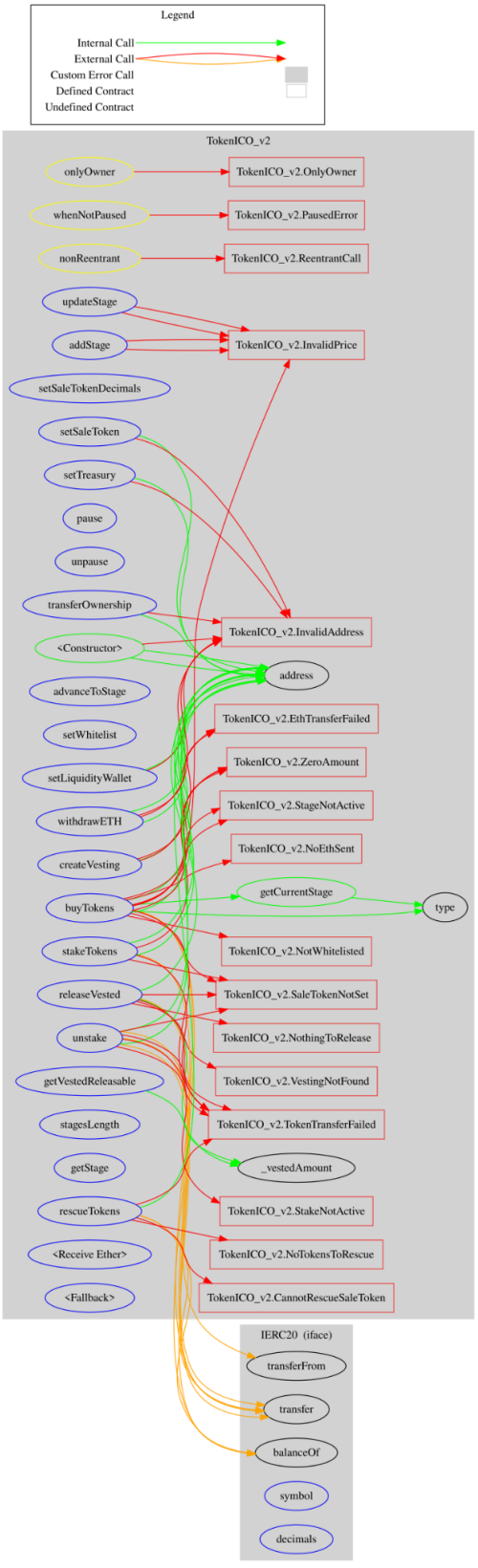
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
	transfer	External	✓	-
	balanceOf	External		-
	transferFrom	External	✓	-
	symbol	External		-
	decimals	External		-
TokenICO_v2	Implementation			
		Public	✓	-
	setSaleToken	External	✓	onlyOwner
	setSaleTokenDecimals	External	✓	onlyOwner
	setTreasury	External	✓	onlyOwner
	setLiquidityWallet	External	✓	onlyOwner
	pause	External	✓	onlyOwner
	unpause	External	✓	onlyOwner
	addStage	External	✓	onlyOwner
	updateStage	External	✓	onlyOwner
	advanceToStage	External	✓	onlyOwner
	getCurrentStage	Public		-
	setWhitelist	External	✓	onlyOwner

	buyTokens	External	Payable	whenNotPaused nonReentrant
	createVesting	External	✓	onlyOwner
	releaseVested	External	✓	nonReentrant
	_vestedAmount	Internal		
	stakeTokens	External	✓	whenNotPaused nonReentrant
	unstake	External	✓	whenNotPaused nonReentrant
	transferOwnership	External	✓	onlyOwner
	withdrawETH	External	✓	onlyOwner nonReentrant
	rescueTokens	External	✓	onlyOwner nonReentrant
	stagesLength	External		-
	getStage	External		-
	getVestedReleasable	External		-
		External	Payable	-
		External	Payable	-

Inheritance Graph

Flow Graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
TokenICO_v2
onlyOwner
TokenICO_v2.OnlyOwner
whenNotPaused
TokenICO_v2.PausedError
nonReentrant
TokenICO_v2.ReentrantCall
updateStage
addStage
TokenICO_v2.InvalidPrice
setSaleTokenDecimals
setSaleToken
setTreasury
pause
unpause
transferOwnership
TokenICO_v2.InvalidAddress
<Constructor>
address
advanceToStage
TokenICO_v2.EthTransferFailed
setWhitelist
TokenICO_v2.ZeroAmount
setLiquidityWallet
TokenICO_v2.StageNotActive
withdrawETH
TokenICO_v2.NoEthSent
createVesting
getCurrentStage
type
buyTokens
stakeTokens
TokenICO_v2.NotWhitelisted
releaseVested
TokenICO_v2.SaleTokenNotSet
unstake
TokenICO_v2.NothingToRelease
getVestedReleasable
TokenICO_v2.VestingNotFound
stagesLength
TokenICO_v2.TokenTransferFailed
getStage
_vestedAmount
rescueTokens
TokenICO_v2.StakeNotActive
<Receive Ether>
TokenICO_v2.NoTokensToRescue
<Fallback>
TokenICO_v2.CannotRescueSaleToken
IERC20 (iface)
transferFrom
transfer
balanceOf
symbol
decimals

Summary

The Sqwad contract implements a presale, staking and vesting mechanism. This audit investigates security issues, business logic concerns and potential improvements.

The team has acknowledged all findings.

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