Cyberscope
A TAC Security Company

Audit Report
Tectum Cash Token

October 2025

Network	ETH
Address	0xE9db0A36877dE86EfD134F87CAF3Db29f90b275c
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

Contract Name	TCT
Compiler Version	v0.8.30+commit.73712a01
Optimization	200 runs
Explorer	https://etherscan.io/address/0xe9db0a36877de86efd134f87caf3db29f90b275c
Address	0xe9db0a36877de86efd134f87caf3db29f90b275c
Network	ETH
Symbol	TCT
Decimals	8
Total Supply	200.000.000

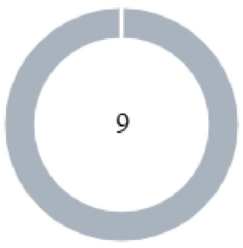
Audit Updates

Initial Audit	07 Oct 2025 https://github.com/cyberscope-io/audits/blob/main/6-tct/v1/audit.pdf
Corrected Phase 2	20 Oct 2025

Source Files

Filename	SHA256
TCT.sol	478e211f72e36357f319387741f61731c98f92e849647fc4751d781f001f12a8

Findings Breakdown



● Critical	0
● Medium	0
● Minor / Informative	9

Severity	Unresolved	Acknowledged	Resolved	Other
● Critical	0	0	0	0
● Medium	0	0	0	0
● Minor / Informative	9	0	0	0

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	CPVA	Cancelled Proposal Voting Acceptance	Unresolved
●	CCR	Contract Centralization Risk	Unresolved
●	IC	Insufficient Check	Unresolved
●	MT	Mints Tokens	Unresolved
●	PVAC	Proposer Voting Acceptance Concern	Unresolved
●	ST	Stops Transactions	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L18	Multiple Pragma Directives	Unresolved

CPVA - Cancelled Proposal Voting Acceptance

Criticality	Minor / Informative
Location	TCT.sol#L6148
Status	Unresolved

Description

The contract allows users to vote on a cancelled proposal. Users that vote on a canceled proposal will not be able to vote on another proposal unless the `lockedUntil` period ends.

Shell
```
function _castVote(uint256 proposalId, address
voter, bool support) private nonReentrant
validProposal(proposalId) {
    if (lockedUntil[voter] > block.timestamp &&
activeProposal[voter] != proposalId) {
        revert TokensLocked();
    }
    ...
}
```

Recommendation

The team is recommended to restrict users from voting on cancelled proposals.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	TCT.sol#L6130
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, the contract declares the `recipient` of token proposals to the owner's address, creating a centralization risk where the owner has full control over the distribution and use of minted tokens, leaving token holders dependent on a single entity's discretion.

Shell
```
function propose(uint256 amount, string calldata
description) external returns (uint256) {
     ...
     address recipient = owner();
     ...
     proposal.recipient = recipient;
     ...

}
```

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.

IC - Insufficient Check

Criticality	Minor / Informative
Location	TCT.sol#L6093
Status	Unresolved

Description

The `_castVote` function prevents users from transferring tokens after they have voted on a proposal; however, this restriction may be ineffective since voting power can be delegated between users resulting in token holders that are still able to transfer their tokens, potentially bypassing the intended limitation.

```
Shell
function _update(address from, address to, uint256
amount) internal override(ERC20, ERC20Votes) {
    if (from != address(0) && lockedUntil[from] >
block.timestamp) {
        uint256 proposalId = activeProposal[from];
        if (proposalId == 0 ||
!proposals[proposalId].cancelled) {
            revert TokensLocked();
        }
        lockedUntil[from] = 0;
        activeProposal[from] = 0;
    }
    super._update(from, to, amount);

}
```

Recommendation

The team should take into consideration that voting power can be delegated and make the necessary changes so that `_update` works as intended.

MT - Mints Tokens

Criticality	Minor / Informative
Location	TCT.sol#L6205
Status	Unresolved

Description

The contract allows token holders to vote on proposals to mint tokens to the contract's owner. Although minting is capped by `MAX_SUPPLY` of `1_000_000_000e8`, the execution of such proposals can still lead to significant token inflation.

Shell
```
function execute(uint256 proposalId) external
nonReentrant validProposal(proposalId) {
   ...
   _mint(proposal.recipient, proposal.amount);
   ...

}
```

Recommendation

The team is recommended to consider the feasibility of implementing additional safeguards to protect the token economy and maintaining stakeholder trust.

PVAC - Proposer Voting Acceptance Concern

Criticality	Minor / Informative
Location	TCT.sol#L6104,6148
Status	Unresolved

Description

Proposal creators are able to delegate their voting power to other addresses they own in order to vote. This introduces a centralization risk where a proposer holding a majority of tokens could unilaterally create, vote on, and execute proposals without input from other users.

```Shell
function propose(uint256 amount, string calldata
description) external returns (uint256)

function castVote(uint256 proposalId, bool
support) external


function execute(uint256 proposalId) external
nonReentrant validProposal(proposalId)
```

Recommendation

The team is recommended to evaluate if refining the governance mechanism by restricting proposers from voting on their own proposals could effectively reduce centralization risks and promote broader community participation.

ST - Stops Transactions

Criticality	Minor / Informative
Location	TCT.sol#L6093
Status	Unresolved

Description

The contract disables token transfers from users that have voted in a proposal until the voting period ends, preventing users from transferring tokens that are actively contributing to governance decisions.

Shell

```
function _update(address from, address to, uint256
amount
) internal override(ERC20, ERC20Votes) {
    if (from != address(0) && lockedUntil[from] >
block.timestamp) {
        uint256 proposalId = activeProposal[from];
        if (proposalId == 0 ||
!proposals[proposalId].cancelled) {
            revert TokensLocked();
        }
        lockedUntil[from] = 0;
        activeProposal[from] = 0;
    }
    super._update(from, to, amount);
}
```

Recommendation

The team is recommended to evaluate whether the lock mechanism could impact token liquidity and user participation, consider implementing a clear mechanism to notify users of remaining lock durations, and assess alternative designs such as allowing partial transfers or delegation to maintain flexibility while preserving the integrity of the voting process.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	TCT.sol#L2401,2404,2416,2420,2421,2422,2423,2424,2425,2431,2496,2498
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

Shell
```
denominator := div(denominator, twos)
...
low := div(low, twos)
...
uint256 inverse = (3 * denominator) ^ 2;
...
inverse *= 2 - denominator * inverse;
...
result = low * inverse;
...
uint256 quotient = gcd / remainder;
...
(gcd, remainder) = (remainder, gcd - remainder *
quotient);
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	TCT.sol#L818,986,2153,2185,2200,2235,2251,2264,2386,2570,2620,2807,3030,3096,3450,3466,3503,3545,3563,3641,3650,3659,3668,3677,3686,3695,3704,3713,3786,4476,4679,4882,5922,5951
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
Shell
assembly ("memory-safe") {
    ...
}

...
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	TCT.sol#L10,92,120,150,315,622,663,756,939,1000,2164,2915,2985,3477,3579,3724,3848,3879,4040,4090,4168,4231,4251,4263,4894,5028,5283,5368,5470,5539,5619,5627,5655,5663,5751,5963
Status	Unresolved

Description

If the contract includes multiple conflicting pragma directives, it may produce unexpected errors. To avoid this, it's important to include the correct pragma directive at the top of the contract and to ensure that it is the only pragma directive included in the contract.

Shell
```
pragma solidity >=0.4.16;
pragma solidity >=0.6.2;
pragma solidity ^0.8.20;

pragma solidity >=0.8.4;
```

Recommendation

It is important to include only one pragma directive at the top of the contract and to ensure that it accurately reflects the version of Solidity that the contract is written in.

By including all required compiler options and flags in a single pragma directive, the potential conflicts could be avoided and ensure that the contract can be compiled correctly.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
TCT	Implementation	ERC20, ERC20Burnable, ERC20Permit, ERC20Votes, Ownable2Step, ReentrancyGuard		
		Public	✓	ERC20 ERC20Permit Ownable
	decimals	Public		-
	_update	Internal	✓	
	propose	External	✓	-
	castVote	External	✓	-
	_castVote	Private	✓	nonReentrant validProposal
	execute	External	✓	nonReentrant validProposal
	cancel	External	✓	validProposal
	_proposalSucceeded	Private		
	state	External		-
	hasVoted	External		-
	proposalDescription	External		-
	recoverTokens	External	✓	onlyOwner nonReentrant
	burn	Public	✓	-

	burnFrom	Public	✓	-
	nonces	Public		-

Inheritance Graph

The inheritance graph of Tectum Cash Token can be found here:

tectum_cash_token_inheritance_graph.png

Flow Graph

The flow graph of Tectum Cash Token can be found here:

tectum_cash_token_flow_graph.png

Summary

Tectum Cash Token contract implements a token 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