



Cyberscope

A *TAC Security* Company

Audit Report

Tectum Cash Token

October 2025

Network ETH

Address 0x2d3009ed2b618e660b1632e8820e9b179003d189

Audited by © cyberscope

Table of Contents

Table of Contents	1
Risk Classification	3
Review	4
Audit Updates	4
Source Files	4
Findings Breakdown	5
Diagnostics	6
ICF - Inconsistent Cancel Functionality	6
Description	7
Recommendation	8
VPD - Voting Power Discrepancy	9
Description	9
Recommendation	9
CO - Code Optimization	10
Description	10
Recommendation	11
CCR - Contract Centralization Risk	12
Description	12
Recommendation	13
IC - Insufficient Check	14
Description	14
Recommendation	15
MT - Mints Tokens	16
Description	16
Recommendation	16
PVAC - Proposer Voting Acceptance Concern	17
Description	17
Recommendation	17
ST - Stops Transactions	18
Description	18
Recommendation	18
L13 - Divide before Multiply Operation	19
Description	19
Recommendation	19
L17 - Usage of Solidity Assembly	20
Description	20
Recommendation	21
L18 - Multiple Pragma Directives	22
Description	22

Recommendation	22
L19 - Stable Compiler Version	23
Description	23
Recommendation	23
Functions Analysis	24
Inheritance Graph	26
Flow Graph	27
Summary	28
Disclaimer	29
About Cyberscope	30

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/0x2d3009ed2b618e660b1632e8820e9b179003d189
Address	0x2d3009ed2b618e660b1632e8820e9b179003d189
Network	ETH
Symbol	TCT
Decimals	8
Total Supply	200.000.000

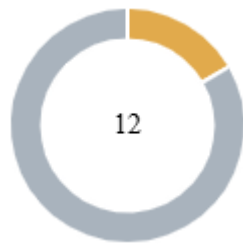
Audit Updates

Initial Audit	07 Oct 2025
----------------------	-------------

Source Files

Filename	SHA256
TCT.sol	f78aa38f15a43ef2d778ffbf5cbec4ad9ead9bb566c71293fad0abba7488d0d

Findings Breakdown



● Critical	0
● Medium	2
● Minor / Informative	10

Severity	Unresolved	Acknowledged	Resolved	Other
● Critical	0	0	0	0
● Medium	2	0	0	0
● Minor / Informative	10	0	0	0

Diagnostics

● Critical
 ● Medium
 ● Minor / Informative

Severity	Code	Description	Status
●	ICF	Inconsistent Cancel Functionality	Unresolved
●	VPD	Voting Power Discrepancy	Unresolved
●	CO	Code Optimization	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
●	L19	Stable Compiler Version	Unresolved

ICF - Inconsistent Cancel Functionality

Criticality	Medium
Location	TCT.sol#L6209
Status	Unresolved

Description

The `proposer` address of a proposal can cancel it at any time as long as it has not been executed or already cancelled. However, this introduces several inconsistencies and potential governance issues:

- Users may still be able to vote on a cancelled proposal
- The proposer can monitor voting results and cancel proposals that are not in their favor
- The proposer could front-run the execution of a proposal to cancel it
- After cancellation, users are blocked from voting on new proposals until the original proposal period ends, limiting participation.

Shell

```
function cancel(uint256 proposalId) external
validProposal(proposalId) {
    ProposalCore storage proposal =
    proposals[proposalId];
    if (proposal.executed || proposal.cancelled)
    revert InvalidProposal();
    if (msg.sender != proposal.proposer) revert
    Unauthorized();
    proposal.cancelled = true;
    emit ProposalCancelled(proposalId);
}
```


Recommendation

The team is recommended to refine the cancellation mechanism by preventing votes on cancelled proposals, restricting proposers from cancelling proposals based on interim results, implementing safeguards against front-running, and ensuring that user voting rights are immediately restored after a cancellation to maintain fairness and active participation in governance.

VPD - Voting Power Discrepancy

Criticality	Medium
Location	TCT.sol#L6127
Status	Unresolved

Description

The `propose` function checks the user's balance from the previous block to determine eligibility for creating a proposal but sets the snapshot block to the current block number. As a result, voters' power is measured based on the block when the proposal was created rather than the previous block, creating a discrepancy between the proposer's and voters' recorded balances and potentially allowing manipulation of the voting outcome.

```
Shell
function propose(address recipient, uint256
amount, string calldata description) external
returns (uint256) {
    ...
    proposal.snapshotBlock =
uint112(block.number);
    ...
}
```

Recommendation

The team is recommended to set the snapshot block to a previous block instead of the current one at the time of proposal creation, ensuring that temporary balance manipulations cannot affect the voting results and preserving the integrity of the governance process.

CO - Code Optimization

Criticality	Minor / Informative
Location	TCT.sol#L6104,6109
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

Specifically, in the `propose` function the user is able to add the address of the recipient as input to the function. However inside the function there is a restriction that revert the function if the `recipient` is not the `owner()` of the contract. Since only one owner can exist at a time the contract should directly access it from storage without the need of an external input.

Shell

```
function propose(address recipient, uint256
amount, string calldata description) external
returns (uint256) {
    ...
    if (recipient != owner()) revert
RecipientNotOwner();
    ...
}
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	TCT.sol#L6109
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 restricts 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(address recipient, uint256
amount, string calldata description) external
returns (uint256) {
    ...
    if (recipient != owner()) revert
RecipientNotOwner();
    ...
}
```

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#L6096,6163
Status	Unresolved

Description

The `_castVote` function prevents the current owner of the contract from voting, but this check is insufficient because the owner could temporarily transfer ownership to another address, cast a vote, and then reclaim ownership, effectively bypassing the restriction.

Shell

```
function _castVote(uint256 proposalId, address voter,
bool support) private nonReentrant
validProposal(proposalId) {
    ...
    if (voter == owner()) {
        revert OwnerCannotVote();
    }
    ...
}
```

Furthermore, the contract may prevent users from transferring tokens after they have voted on a proposal; however, this restriction may be ineffective since voting power can be delegated to others 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) {
        revert TokensLocked();
    }
    super._update(from, to, amount);
}
```

Recommendation

The team is recommended to implement a more robust mechanism to prevent owner influence on governance, such as tracking historical ownership during voting periods, permanently excluding current and recent owners from voting, or introducing time-locked restrictions on ownership transfers that coincide with active proposals.

MT - Mints Tokens

Criticality	Minor / Informative
Location	TCT.sol#L6204
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#L6103,6147
Status	Unresolved

Description

The contract allows the address that created a proposal to vote on it, introducing 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(address recipient, 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#L6096
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) {
        revert TokensLocked();
    }
    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#L2397,2400,2412,2416,2417,2418,2419,2420,2421,2427,2492,2494
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
inverse *= 2 - denominator * inverse;
...
```

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#L814,982,2149,2181,2196,2231,2247,2260,2382,2566,2616,2803,3026,3092,3446,3462,3499,3541,3559,3637,3646,3655,3664,3673,3682,3691,3700,3709,3782,4472,4675,4878,5918,5947
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") {
    r := mload(add(signature, 0x20))
    s := mload(add(signature, 0x40))
    v := byte(0, mload(add(signature, 0x60)))
}

...
assembly ("memory-safe") {
    mstore(0x00, 0x4e487b71)
    mstore(0x20, code)
    revert(0x1c, 0x24)
}

...
```

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#L6,88,116,146,311,618,659,752,935,996,2160,2911,2981,3473,3575,3720,3844,3875,4036,4086,4164,4227,4247,4259,4890,5024,5279,5364,5466,5535,5615,5623,5651,5659,5747,5959
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 >=0.8.4;
pragma solidity >=0.6.2;
pragma solidity ^0.8.20;
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	TCT.sol#L311,5535
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.

```
Shell  
solidity ^0.8.20;
```

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
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