Tea token

Security Review

Cantina Managed review by:
M4rio.eth, Security Researcher
Sujith Somraaj, Security Researcher

December 31, 2024

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Tea is a decentralized protocol for open-source developers to capture the value they create.

On Dec 31st the Cantina team conducted a review of tea-token on commit hash 22519ec4. The team identified a total of **9** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 0
- Medium Risk: 0
- Low Risk: 2
- Gas Optimizations: 0
- Informational: 7

3 Findings

3.1 Low Risk

3.1.1 Zero amount minting can lock minting capability for a year

Severity: Low Risk

Context: MintManager.sol#L53

Description: The `mintTo` function within the `MintManager.sol` contract enables zero-amount mints that refresh the minting cooldown period. This capability can obstruct valid minting activities by resetting the cooldown timer without minting any tokens.

Recommendation: Consider implementing a minimum threshold for minting operations:

```
function mintTo(address _account, uint256 _amount) public onlyOwner {
     require(mintPermittedAfter <= block.timestamp, "MintManager: minting not permitted yet");

     uint256 minAmount = (tea.totalSupply() * MIN_MINT_THRESHOLD) / DENOMINATOR;
     require(_amount >= minAmount, "MintManager: amount below minimum threshold");
     require(_amount <= (tea.totalSupply() * MINT_CAP) / DENOMINATOR, "MintManager: mint amount exceeds cap");

     mintPermittedAfter = block.timestamp + MINT_PERIOD;
     tea.mintTo(_account, _amount);
}
```

Tea: Acknowledged.

Cantina Managed: Acknowledged.

3.1.2 Minting cap can be bypassed by changing `MintManager`

Severity: Low Risk

Context: MintManager.sol#L63

Description: The minting restrictions, which include a 2% cap and a timelock, are enforced in the `MintManager.sol` contract instead of within the `Tea.sol` token contract.

Because the `MintManager` can be replaced via the upgrade function, the restrictions can be circumvented if ownership is transferred to a different mint manager without these restrictions or to an externally owned account (EOA).

The underlying issue arises from the design choice to place minting restrictions in a separate, upgradeable contract rather than embedding them directly in the token contract.

Recommendation: Consider placing the minting restrictions directly within the Tea contract to ensure strict enforcement.

Tea: Acknowledged. The intention is to mirror the OP setup.

Cantina Managed: Acknowledged.

3.2 Informational

3.2.1 Missing NatSpec for some functions and variables in `Tea.sol`

Severity: Informational

Context: Tea.sol#L34, Tea.sol#L54, Tea.sol#L62

Description: The `Tea` token is missing a `NatSpec` comment on the `mintTo` and `burnFrom` functions and `totalMinted` state variable:

```
uint256 public totalMinted;

function mintTo(address account, uint256 value) public {
    // ...
}

function burnFrom(address account, uint256 value) public override {
    // ...
}
```

Recommendation: Consider adding NatSpec comments to this function:

```
/// @notice Total number of tokens minted, including burned tokens
uint256 public totalMinted;

/// @notice Mints new tokens to `account` (only callable by the owner).
/// @dev Increments `totalMinted`.
/// @param account The address to receive minted tokens.
/// @param value   The amount of tokens to be minted.
function mintTo(address account, uint256 value) public {
    // ...
}

/// @inheritdoc ERC20Burnable
function burnFrom(address account, uint256 value) public override {
   // ...
}
```

Tea: Fixed in commit 4ffd816f.

Cantina Managed: Fixed.

3.2.2 The comment on `MINT_CAP` is misleading

Severity: Informational

Context: MintManager.sol#L24

Description: The `MINT_CAP` comment says that the value is a fixed point number with 4 decimals:

```
/// @notice The amount of tokens that can be minted per year.
   ///         The value is a fixed point number with 4 decimals.
   uint256 public constant MINT_CAP = 20; // 2%
```

In fact it's with 3 decimals because the `DENOMINATOR` is 1000.

Recommendation: Consider either changing the comments or the denomination to `10_000`. If denomination is changed then consider changing the `MINT_CAP` to 200 as well.

Tea: Fixed in commit f094d038.

Cantina Managed: Fixed.

3.2.3 The 2% minting per year can be delayed if it's not called at the exact time

Severity: Informational

Context: MintManager.sol#L57

Description: The `MintManager` allows increasing the supply by a maximum of 2% once every 356 days. To maintain a consistent 2% increase every 365 days, the owner must call `mintTo` exactly every 356 days. If this is not done, the next minting period will be delayed because the next permitted mint time is set as `block.timestamp + MINT_PERIOD`:

```
function mintTo(address _account, uint256 _amount) public onlyOwner {
    require(mintPermittedAfter <= block.timestamp, "MintManager: minting not permitted yet");
    require(_amount <= (tea.totalSupply() * MINT_CAP) / DENOMINATOR, "MintManager: mint amount exceeds cap");

    mintPermittedAfter = block.timestamp + MINT_PERIOD;
    tea.mintTo(_account, _amount);
}
```

Recommendation: Consider whether this behavior is intentional. If not, adjust the `mintPermittedAfter` calculation to add to its current value instead of using `block.timestamp`:

```
mintPermittedAfter = mintPermittedAfter + MINT_PERIOD;
```

Tea: Acknowledged, 2% of current supply is intended to be a max.

Cantina Managed: Acknowledged.

3.2.4 Increase test coverage

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: There is less than complete test coverage of key contracts under review. Adequate test coverage and regular reporting are essential to ensure the codebase works as intended. Insufficient code coverage may lead to unexpected issues and regressions.

Filename
MintManager.sol
Tea.sol
TokenDeploy.sol

Recommendation: Add to test coverage, ensuring all execution paths/branches are covered. The missing test cases are added as follows:

- Test Transfer Functionality:

```
function test_transfer_functionality() public {
    vm.warp(block.timestamp + 365 days);

    // Mint some tokens to alice
    vm.prank(initialGovernor.addr);
    mintManager.mintTo(alice.addr, 100);

    // Test transfer
    vm.prank(alice.addr);
    tea.transfer(bob.addr, 50);

    assertEq(tea.balanceOf(alice.addr), 50);
    assertEq(tea.balanceOf(bob.addr), 50);
}
```

- Test Approve and TransferFrom:

```
function test_approve_and_transferFrom() public {
    vm.warp(block.timestamp + 365 days);

    // Mint some tokens to alice
    vm.prank(initialGovernor.addr);
    mintManager.mintTo(alice.addr, 100);

    // Alice approves Bob to spend 30 tokens
    vm.prank(alice.addr);
    tea.approve(bob.addr, 30);

    // Bob transfers 20 tokens from Alice to himself
    vm.prank(bob.addr);
    tea.transferFrom(alice.addr, bob.addr, 20);

    assertEq(tea.balanceOf(alice.addr), 80);
    assertEq(tea.balanceOf(bob.addr), 20);
    assertEq(tea.allowance(alice.addr, bob.addr), 10);
}
```

- Test Burn Functionality:

```
function test_burn_succeed() public {
    vm.warp(block.timestamp + 365 days);

    vm.prank(initialGovernor.addr);
    mintManager.mintTo(alice.addr, 1);

    vm.prank(alice.addr);
    tea.approve(address(this), 1);

    tea.burnFrom(alice.addr, 1);

    assertEq(tea.totalSupply(), tea.INITIAL_SUPPLY());
    assertEq(tea.totalMinted(), tea.INITIAL_SUPPLY() + 1);
    assertEq(tea.balanceOf(alice.addr), 0);
}
```

- Test Zero Address Transfers:

```
function test_zero_address_transfers() public {
    vm.warp(block.timestamp + 365 days);

    vm.prank(initialGovernor.addr);
    mintManager.mintTo(alice.addr, 100);

    vm.prank(alice.addr);
    vm.expectRevert(abi.encodeWithSelector(IERC20Errors.ERC20InvalidReceiver.selector, address(0)));
    tea.transfer(address(0), 50);
}
```

- Test Mint to Zero Address:

```
function test_mint_toZeroAddress_reverts() external {
    vm.warp(block.timestamp + 365 days);
    vm.prank(initialGovernor.addr);
    vm.expectRevert(abi.encodeWithSelector(IERC20Errors.ERC20InvalidReceiver.selector, address(0)));
    mintManager.mintTo(address(0), 100);
}
```

- Test Multiple Mints Within Period:

```
function test_mint_multipleMints_withinPeriod_reverts() external {
    // First mint after 1 year
    vm.warp(block.timestamp + 365 days);
    vm.startPrank(initialGovernor.addr);

    // Mint 1% first
    uint256 onePercent = (tea.totalSupply() * 10) / mintManager.DENOMINATOR();
    mintManager.mintTo(initialGovernor.addr, onePercent);

    // Try to mint another 1% - should fail as the mint period has not elapsed
    uint256 onePointFivePercent = (tea.totalSupply() * 10) / mintManager.DENOMINATOR();
    vm.expectRevert("MintManager: minting not permitted yet");
    mintManager.mintTo(initialGovernor.addr, onePointFivePercent);
    vm.stopPrank();
}
```

- Test Minting at Period Boundary:

```
function test_mint_exactlyAtPeriodBoundary_reverts() external {
    uint256 ts = block.timestamp;
    // First mint after 1 year
    vm.warp(ts + 365 days);
    vm.prank(initialGovernor.addr);
    mintManager.mintTo(initialGovernor.addr, 100);

    // Try minting exactly at mintPermittedAfter (should fail)
    vm.warp(ts + 365 days + mintManager.MINT_PERIOD() - 1);
    vm.prank(initialGovernor.addr);
    vm.expectRevert("MintManager: minting not permitted yet");
    mintManager.mintTo(initialGovernor.addr, 100);
}
```

- Test Minting at Cap Limit:

```
function test_mint_exactlyAtCap_succeeds() external {
    vm.warp(block.timestamp + 365 days);
    vm.startPrank(initialGovernor.addr);

    // Calculate exact 2% of total supply
    uint256 exactCap = (tea.totalSupply() * mintManager.MINT_CAP()) / mintManager.DENOMINATOR();
    mintManager.mintTo(initialGovernor.addr, exactCap);

    // Verify balance increased by exactly 2%
    assertEq(
        tea.balanceOf(initialGovernor.addr),
        tea.INITIAL_SUPPLY() + exactCap
    );
    vm.stopPrank();
}
```

- Test Upgrade from Owner (with more assertions):

```
function test_upgrade_fromOwner_succeeds() external {
    // Upgrade to new mintManager
    vm.prank(initialGovernor.addr);
    mintManager.upgrade(alice.addr);

    // Check pending state
    assertEq(tea.owner(), address(mintManager));
    assertEq(tea.pendingOwner(), alice.addr);

    vm.prank(alice.addr);
    tea.acceptOwnership();

    // New manager is alice.addr
    assertEq(tea.owner(), alice.addr);
    assertEq(tea.pendingOwner(), address(0));
}
```

The above tests use `IERC20Errors` from `@openzeppelin/contracts/interfaces/draft-IERC6093.sol`; hence, import them appropriately.

Tea: Fixed in commit f79268d7.

Cantina Managed: Fixed. Other branch of TokenDeploy.sol#L53 can never be covered ig.

3.2.5 Unnecessary `virtual` keyword in `_update` function

Severity: Informational

Context: Tea.sol#L48

Description: The `_update` function in `Tea.sol` is marked as **virtual**, but there is no apparent need for further overriding.

Recommendation: Consider removing the `virtual` keyword unless there's specific need for further inheritence.

Tea: Fixed in commit 16cead01.

Cantina Managed: Fixed.

3.2.6 External functions are declared `public`

Severity: Informational

Context: Tea.sol#L54

Description: Some functions in the target contracts are currently declared as `public` but are only called externally. While there are no gas savings from optimization, declaring functions as external enhances code quality.

- `Tea.sol`

```
function mintTo(address account, uint256 value) public {
    // ...
}
```

- `TokenDeploy.sol`

```
function deploy(bytes32 salt, bytes32 salt2) external {
    // ...
}
```

- `MintManager.sol`

```
function mintTo(address _account, uint256 _amount) public onlyOwner {
    // ...
}


function upgrade(address _newMintManager) public onlyOwner {
    // ...
}
```

Recommendation: Change the function visibility modifier from `public` to `external`, if its not intended to be called internally.

Tea: Fixed in commit bfb861a8.

Cantina Managed: Fixed.

3.2.7 Remove redundant `burnFrom` implementation

Severity: Informational

Context: Tea.sol#L62

Description: The `burnFrom` function in the `Tea.sol` contract is an exact duplicate of the implementation from `ERC20Burnable.sol` and can be safely removed.

```
// Current Tea contract
function burnFrom(address account, uint256 value) public override {
    if (account != msg.sender) _spendAllowance(account, msg.sender, value);
    _burn(account, value);
}

// From ERC20Burnable
function burnFrom(address account, uint256 value) public virtual {
    _spendAllowance(account, _msgSender(), value);
    _burn(account, value);
}
```

Recommendation: Consider removing the burnFrom function from the Tea contract entirely, as it's already inherited from `ERC20Burnable`.

Tea: Fixed in commit 9af4fcf2.

Cantina Managed: Fixed.