SALUS SECURITY

AUG 2025

CODE SECURITY ASSESSMENT

DGC

Overview

Project Summary

- Name: Decentral GPT Token
- Platform: EVM-compatible chains
- Language: Solidity
- Address:
  - DeepBrainChain:
    - proxy: 0x8E5e4a4d8aE3741DA073303e492B73cb913fb72D
    - Implementation:0x48f23A5A354AF81e9315a78A5081b60CabB2Bf6B
  - Binance Smart Chain:
    - proxy: 0x9cfAE8067322394e34E6b734c4a3F72aCC4a7Fe5
    - Implementation: 0x7A83045A1A0450870a4B02c0feA01504B077e2e2
- Repository:
  - https://github.com/AIDecentralGPT/DecentralGPTTokenContract
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Decentral GPT Token
Version	v5
Type	Solidity
Dates	Aug 04 2025
Logs	Jul 29 2025; Jul 31 2025; Aug 04 2025; Aug 19 2025; Aug 20 2025

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	1
Total Low-Severity issues	3
Total informational issues	3
Total	7

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	The balance may be less than the lock balance	Medium	Business Logic	Resolved
2	The function name does not match its actual logic	Low	Business Logic	Resolved
3	Different variable types may result in variable size mismatches.	Low	Code Quality	Resolved
4	Centralization risk	Low	Centralization	Acknowledged
5	Redundant Code	Informational	Redundancy	Resolved
6	Missing two-step transfer ownership pattern	Informational	Business Logic	Acknowledged
7	Use of floating pragma	Informational	Configuration	Acknowledged

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. The balance may be less than the lock balance

Severity: Medium	Category: Business Logic
Target: - All	

Description

The `transfer` and `transferFrom` functions both allow transfers to the zero address; however, they do not update the lock data accordingly. If a user's tokens are burned in this way (i.e., transferred to the zero address), their balance may become less than their locked amount, which can lead to functions `canTransferAmount` and `getAvailableAmount` to be unavailable.

contracts/Token(bsc).sol:L121 - L143
contracts/Token(dbc).sol:L127 - L149

```
function transfer(address to, uint256 amount) public virtual override returns (bool) {
    if (to == address(0) || amount == 0) {
        return super.transfer(to, amount);
    }
    ...
}

function transferFrom(address from, address to, uint256 amount) public virtual override
returns (bool) {
    if (to == address(0) || amount == 0) {
        return super.transferFrom(from, to, amount);
    }
    ...
}
```

The `canTransferAmount` and `getAvailableAmount` functions are calculated as follows:

```
uint256 lockedAmount = calculateLockedAmount(caller);
uint256 total = balanceOf(caller);
uint256 availableAmount = total - lockedAmount;
```

Recommendation

It is recommended to update the user's corresponding `lockInfo` when tokens are transferred to the zero address, or to adopt an alternative mechanism to properly track and reflect such operations.

Status

This issue has been resolved by the team at commit 877829b and 2830c89.

2. The function name does not match its actual logic

Severity: Low		Category: Business Logic
Target: - All		

Description

The function names `disableLockPermanently` and `enableLockPermanently` suggest that these are one-time, irreversible operations. However, both functions can be called multiple times.

contracts/Token(bsc).sol:L84 - L92
contracts/Token(dbc).sol:L90 - L98

```
function disableLockPermanently() external onlyOwner {
...
}

function enableLockPermanently() external onlyOwner {
...
}
```

Recommendation

It is recommended to either rename the functions to better reflect their behavior or modify the logic to ensure the operations are indeed permanent.

Status

This issue has been resolved by the team at commit 95927cb and c09e885.

3. Different variable types may result in variable size mismatches.

Severity: Low	Category: Code Quality
Target: - All	

Description

In the `Token` contract, the type of the `lockLimit` variable is `uint256`, so theoretically, the maximum length of the `LockInfo` array should also be this much. However, in the `getLockAmountAndUnlockAt()` function, the type of the `index` parameter is `uint16`.

contracts/Token(bsc).sol:L189 - L193
contracts/Token(dbc).sol:L211 - L215

```
function getLockAmountAndUnlockAt(address caller, uint16 index) public view returns
(uint256, uint256) {
    require(index < walletLockTimestamp[caller].length, "Index out of range");
    LockInfo memory lockInfo = walletLockTimestamp[caller][index];
    return (lockInfo.lockedAmount, lockInfo.unlockAt);
}
```

If the length of the `LockInfo` array exceeds the maximum value of uint16, this will cause the function to be unable to read the portion of the array with indices greater than the maximum value of uint16, although the probability of exceeding 2**16 is low, it is still recommended to maintain consistency throughout.

Recommendation

It is recommended to keep variable types consistent.

Status

This issue has been resolved by the team at commit 95927cb and c09e885.

4. Centralization risk

Severity: Low	Category: Centralization
Target: - All	

Description

In the `Token` contract, the `owner` holds special privileges, such as controlling whether locking is enabled, assigning lock data to any address, adding or removing `TransferAdmin` roles, updating `lockLimit`, and performing various other critical operations.

If the owner's private key is compromised, an attacker could exploit these privileges to add large short-term locks for themselves, transfer a significant amount of tokens to themselves, or even upgrade the logic contract to a malicious version—resulting in severe threats to the project and user funds.

If the privileged accounts are plain EOA accounts, this can be worrisome and pose a risk to the other users.

Recommendation

We recommend transferring privileged accounts to multi-sig accounts with timelock governors for enhanced security. This ensures that no single person has full control over the accounts and that any changes must be authorized by multiple parties.

Status

This issue has been acknowledged by the team.

2.3 Informational Findings

5. Redundant Code

Severity: Informational	Category: Redundancy
Target: - All	

Description

Unused code should be removed before deploying the contract to mainnet. We have identified the following code are not being utilized:

contracts/Token(bsc).sol:L36 - L38

```
uint256 public amountToIAO;

mapping(uint256 => uint256) public mintedPerYear;
```

contracts/Token(bsc).sol:L49 - L50

contracts/Token(dbc).sol:L53 - L54

```
event AuthorizedUpgradeSelf(address indexed canUpgradeAddress);
event DisableContractUpgrade(uint256 timestamp);
```

contracts/Token(dbc).sol:L40

```
uint256 public amountToIAO;
```

Recommendation

Consider removing the redundant code.

Status

This issue has been resolved by the team at address [0x1c30c2af4e4c1742d6db75afe85f1673ae2a8794](0x1c30c2af4e4c1742d6db75afe85f1673ae2a8794).

6. Missing two-step transfer ownership pattern

Severity: Informational	Category: Business logic
Target: - All	

Description

The `Token` contract inherits from the `OwnableUpgradeable` contract. This contract does not implement a two-step process for transferring ownership. Thus, ownership of the contract can easily be lost when making a mistake in transferring ownership.

Recommendation

Consider using the Ownable2StepUpgradeable contract from OpenZeppelin instead.

Status

This issue has been acknowledged by the team.

7. Use of floating pragma

Severity: Informational	Category: Configuration
Target: - contracts/Token(dbc).sol	

Description

```
pragma solidity ^0.8.22;
```

The `Token` contract uses a floating compiler version `^0.8.22`.

Using a floating pragma `^0.8.22`statement is discouraged, as code may compile to different bytecodes with different compiler versions. Use a locked pragma statement to get a deterministic bytecode. Also use the latest Solidity version to get all the compiler features, bug fixes and optimizations.

Recommendation

It is recommended to use a locked Solidity version throughout the project. It is also recommended to use the most stable and up-to-date version.

Status

This issue has been acknowledged by the team.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 4802a62 and 8a2d1a7:

File	SHA-1 hash
contracts/Token(bsc).sol	e658bda0b7e243e72d7e2c77a971af49c30ad629
contracts/Token(dbc).sol	176a003bfa298e96c4cc019c93da50fcd3ee254d