Assure DeFi®

THE VERIFICATION GOLD STANDARD

Security Assessment

Blink Galaxy

Date: 05/12/2025

Audit Status: PASS

Audit Edition: Advanced+

Risk Analysis

Vulnerability summary

Classification	Description
🔴 High	High-level vulnerabilities can result in the loss of assets or manipulation of data.
🟠 Medium	Medium-level vulnerabilities can be challenging to exploit, but they still have a considerable impact on smart contract execution, such as allowing public access to critical functions.
🟡 Low	Low-level vulnerabilities are primarily associated with outdated or unused code snippets that generally do not significantly impact execution, sometimes they can be ignored.
🟢 Informational	Informational vulnerabilities, code style violations, and informational statements do not affect smart contract execution and can typically be disregarded.

Executive Summary

According to the Assure assessment, the Customer's smart contract is **Secured.**

Scope

Target Code And Revision

For this audit, we performed research, investigation, and review of the Blink Galaxy contracts followed by issue reporting, along with mitigation and remediation instructions outlined in this report.

Target Code And Revision

Project	Assure
Language	Solidity
Codebase	https://basescan.org/address/0x701C8C09fE9F081f9BE69Ab9AF8291c84c09389B#code
Audit Methodology	Static, Manual

Attacks made to the contract

In order to check for the security of the contract, we tested several attacks in order to make sure that the contract is secure and follows best practices.

Category	Item
Code review & Functional Review	• Compiler warnings. • Race conditions and Reentrancy. Cross-function race conditions. • Possible delays in data delivery. • Oracle calls. • Front running. • Timestamp dependence. • Integer Overflow and Underflow. • DoS with Revert. • DoS with block gas limit. • Methods execution permissions. • Economy model. • Private user data leaks. • Malicious Event log. • Scoping and Declarations. • Uninitialized storage pointers. • Arithmetic accuracy. • Design Logic. • Cross-function race conditions. • Safe Zeppelin module. • Fallback function security. • Overpowered functions / Owner privileges

AUDIT OVERVIEW

1. Governance vote accounting underflow causing DoS on transfers/burns/delegation [Acknowledge ✅]

Issue:

The governance delegation system is inconsistent with token balances when tokens are minted, and this can cause arithmetic underflow in _moveDelegates, which in Solidity ≥ 0.8.x results in a revert.

This creates a Denial-of-Service condition for:

- transfer
- transferFrom
- burn
- delegate (changing delegation)
- delegateBySig

for certain users, effectively freezing their tokens and breaking governance.

Root Cause

Votes are updated only in:

- transfer
- transferFrom
- burn
- _delegate

but not in mint.

```
function mint(address to, uint256 amount) external onlyRole(MINTER_ROLE) {
    require(totalSupply() + amount <= _cap, "Cap exceeded");
    _mint(to, amount);
    emit Minted(msg.sender, to, amount);
}
```

_mint (from OZ's ERC20) will eventually call the overridden _update(from, to, value), but that override does not call _moveDelegates. So delegated voting power is not updated on mint.

The delegation movement logic:

```
function _moveDelegates(address srcRep, address dstRep, uint256 amount) internal {
    if (srcRep != dstRep && amount > 0) {
        if (srcRep != address(0)) {
            uint32 srcRepNum = numCheckpoints[srcRep];
            uint256 srcRepOld = srcRepNum > 0 ? checkpoints[srcRep][srcRepNum -
1].votes : 0;
            uint256 srcRepNew = srcRepOld - amount; // <-- potential underflow
```

```
            _writeCheckpoint(srcRep, srcRepNum, srcRepOld, srcRepNew);
        }

        if (dstRep != address(0)) {
            uint32 dstRepNum = numCheckpoints[dstRep];
            uint256 dstRepOld = dstRepNum > 0 ? checkpoints[dstRep][dstRepNum -
1].votes : 0;
            uint256 dstRepNew = dstRepOld + amount;
            _writeCheckpoint(dstRep, dstRepNum, dstRepOld, dstRepNew);
        }
    }
}
```

In Solidity 0.8., srcRepOld - amount reverts if amount > srcRepOld.

There is no guarantee that srcRepOld >= amount because the vote checkpoints are not updated on mint. This breaks the usual Compound-style invariant that "representative votes always equal the sum of delegated balances".

Failure scenarios:

- User delegates before being minted tokens = transfer/burn broken
- Delegator tries to change delegate after mint = delegation change impossible

Recommendation: You have two main options:

Option 1 = Properly integrate delegation with all balance changes (preferred)

Mimic the standard Compound-style pattern and ensure _moveDelegates is invoked for all balance changes: transfers, mints, and burns.

A robust approach with modern OZ is to hook into _update or _afterTokenTransfer:

Add delegate adjustment in _update:

```
function _update(address from, address to, uint256 value) internal override {
    if (whitelistEnabled) {
        if (from == address(0)) {
            require(isWhitelisted[to], "to not whitelisted");
        } else if (to == address(0)) {
            require(isWhitelisted[from], "from not whitelisted");
        } else {
            require(isWhitelisted[from] && isWhitelisted[to], "transfer not
whitelisted");
        }
    }

    // Capture delegates before balances change
    address fromDelegate = from == address(0) ? address(0) : _delegates[from];
    address toDelegate   = to == address(0) ? address(0) : _delegates[to];

    super._update(from, to, value);

    if (value > 0) {
        if (from != address(0)) {
            _moveDelegates(fromDelegate, address(0), value); // remove from
```

```
fromDelegate
        }
        if (to != address(0)) {
            _moveDelegates(address(0), toDelegate, value);   // add to toDelegate
        }
    }
}
```

Then remove the manual _moveDelegates calls from transfer, transferFrom, and burn to avoid double-counting.

Or, if using _afterTokenTransfer, you can do it there instead; just make sure:

- Mints (from == address(0)) call _moveDelegates(address(0), delegateeOf(to), amount).
- Burns (to == address(0)) call _moveDelegates(delegateeOf(from), address(0), amount).

Option 2 = Add invariant checks and repair logic

If you want minimal code change:

At minimum, protect _moveDelegates from underflow:

```
require(srcRepOld >= amount, "vote underflow");
```

But this only changes failures from “silent math-panic” to explicit revert it does not fix the underlying governance inconsistency that produces srcRepOld < amount.

So, Option 2 is only partial; Option 1 is needed for correct and safe governance behaviour.

Mitigation: The team has acknowledged the issue where delegation vote accounting is not updated on mint, leading to a potential underflow in _moveDelegates and a DoS condition for affected users. At the time of the audit, this risk is being operationally contained through the following controls:

MINTER_ROLE is held exclusively by a multisig deployed at inception; there are no additional minter EOAs or auxiliary minter contracts.

No permissionless or automated minting flows are used.

The team has established and is enforcing an internal policy not to mint directly to wallets that have already delegated. Minting for distribution is performed only to non-delegating treasury/multisig addresses, from which tokens are then distributed via standard transfers.

Related functionality (for example staking vault creation and governance flows) is implemented via dedicated contracts, structurally separating mint operations from user governance/delegation interactions.

If necessary, whitelist gating can be enabled to further restrict potential mint recipients to a controlled set of addresses.

These measures significantly reduce the likelihood of the vulnerable state (minting to an already-delegated account) arising in production.

MEDIUM

---

1. Minted tokens do not grant voting power until further actions [Acknowledge ✅]

Issue:

Even ignoring the underflow/DoS issue, there is a separate governance logic flaw:

When tokens are minted to an account that already has a delegate, the delegate's vote count is not updated.

mint(to, amount) only calls _mint → _update → no _moveDelegates.

The delegate's vote checkpoints are stale until:

- the holder transfers tokens, or
- the holder re-delegates.

That means:

- Delegated voting power can be significantly lower than the sum of delegated balances.
- Governance decisions based on getCurrentVotes / getPriorVotes will be incorrect and potentially manipulable.

Recommendation:

Same as for Finding High: integrate _moveDelegates correctly for mint/burn:

Ensure that all balance changes are reflected in vote checkpoints, not only transfers and manual delegation calls.

After that change, call getCurrentVotes / getPriorVotes during tests in various mint/burn/transfer sequences to confirm invariants like:

Sum of delegates' votes == totalSupply – (tokens held by accounts with no delegate)

LOW

No low issues were found.

INFORMATIONAL

No informational issues were found.

Testing coverage

During the testing phase, custom use cases were written to cover all the logic of contracts. *Check “Annexes” to see the testing code.*

```
tests/test_blink_galaxy.py::test_burn RUNNING
Transaction sent: 0x4e600c25ad1601e27eda666fd724d9c4f7f6c5dcc620f49085043abd49d7d9f1
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 4
  BlinkGalaxyV2.constructor confirmed   Block: 14   Gas used: 2050934 (17.09%)
  BlinkGalaxyV2 deployed at: 0xe0aA552A10d7EC8760Fc6c246D391E698a82dDf9

Transaction sent: 0xf9a95800edf5d4f5e4206eab7bab9960aa14f86bda763b9b96c9f322d22dd947
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 9
  BlinkGalaxyV2.mint confirmed   Block: 15   Gas used: 70631 (0.59%)

Transaction sent: 0xf872a8c0e7fb10e3e3ae197bff95aecac7c743161d4861b83f83c1d3819676d3
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 10
  BlinkGalaxyV2.mint confirmed   Block: 16   Gas used: 55643 (0.46%)

Transaction sent: 0xc55a9395eda17b4110983e26bed366f9141fbcadfcfad816f1d0be7d2a4f3322
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  BlinkGalaxyV2.burn confirmed   Block: 17   Gas used: 38733 (0.32%)

Transaction sent: 0xd5f0cfc89c4f21d635feebcbaf237e54cbdfc0dc75c765b73f296e735667aa13
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  BlinkGalaxyV2.burn confirmed   Block: 18   Gas used: 38733 (0.32%)

Transaction sent: 0xbecceca415662d09a46883930c53f44514ce0092992e02b7e3df29e2f180b9cf
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 11
  BlinkGalaxyV2.setWhitelistEnabled confirmed   Block: 19   Gas used: 44573 (0.37%)

Transaction sent: 0x287a63f6948ab5afc634843ebd5c407afc31917a2722305d538f2541bf2f33de
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  BlinkGalaxyV2.burn confirmed (from not whitelisted)   Block: 20   Gas used: 23526 (0.20%)

tests/test_blink_galaxy.py::test_burn PASSED
tests/test_blink_galaxy.py::test_transfer RUNNING
Transaction sent: 0x86608aa8811f6be37e7a6b9c2ef69aeb46534de2e067d63a2081e6ea54443b8b
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 5
  BlinkGalaxyV2.constructor confirmed   Block: 21   Gas used: 2050934 (17.09%)
  BlinkGalaxyV2 deployed at: 0x6b4BDe1086912A6Cb24ce3dB43b3466e6c72AFd3

Transaction sent: 0xa1d6ff5bfc600ae3a45831f8645a1bae9f016d46f983cea267a201ec73c6900d
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 12
  BlinkGalaxyV2.mint confirmed   Block: 22   Gas used: 70631 (0.59%)

Transaction sent: 0x320ec0a0a9afa981590a8a8b7900480d936c5dcad86ef18d962d7b4a3d0d866c
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 13
  BlinkGalaxyV2.mint confirmed   Block: 23   Gas used: 55643 (0.46%)

Transaction sent: 0xcd6a5af90debc8918dc198b45298521c3ea45113006349ec5c991c417ce36ac6
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  BlinkGalaxyV2.transfer confirmed   Block: 24   Gas used: 38928 (0.32%)

Transaction sent: 0x9fcf091323c9ac801ae379351cff91192e45d93e87624d8d141cf9fabf54b016
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 2
  BlinkGalaxyV2.transfer confirmed   Block: 25   Gas used: 53928 (0.45%)

Transaction sent: 0x7157161ff65d998747948c1b77468032a93ff71470d3d221eb7eb0c7a59497a2
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 14
  BlinkGalaxyV2.setWhitelistEnabled confirmed   Block: 26   Gas used: 44573 (0.37%)

Transaction sent: 0xd818be53cf8f88327d7aa500ebaa1a57f5c89bc91c2b8b551c932eb5d203f4ef
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 3
  BlinkGalaxyV2.transfer confirmed (transfer not whitelisted)   Block: 27   Gas used: 24126 (0.20%)

Transaction sent: 0x0ab1397880ed0bfbf090177c01b7e482c7845803c68c25100fec2613ad91da7b
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 15
  BlinkGalaxyV2.batchSetWhitelisted confirmed   Block: 28   Gas used: 69746 (0.58%)

Transaction sent: 0xc01bf8129da9a2bb59b1401aedd6ba3ea1808614fa61eaf24e4e7ed0252875c9
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 4
  BlinkGalaxyV2.transfer confirmed   Block: 29   Gas used: 40825 (0.34%)

tests/test_blink_galaxy.py::test_transfer PASSED
```

```
tests/test_blink_galaxy.py::test_emergency_withdraw RUNNING
Transaction sent: 0x3e4a63cdfcccda37538fa788c46cd0b527986292a589f2a0df65e3edd389ece6
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 6
  ERC20Mock.constructor confirmed   Block: 30   Gas used: 523834 (4.37%)
  ERC20Mock deployed at: 0x9E4c14403d7d9A8A782044E86a93CAE09D7B2ac9

Transaction sent: 0x9b360e26870d4de56b8b6ed0bac8768bdf4738a91552372333464fba63c538f6
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 7
  BlinkGalaxyV2.constructor confirmed   Block: 31   Gas used: 2050922 (17.09%)
  BlinkGalaxyV2 deployed at: 0xcCB53c9429d32594F404d01fbe9E65ED1DCda8D9

Transaction sent: 0xab4fe8dc75d86f942968698c72990b0562a8e6424146df141196f511d6bdf720
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 16
  BlinkGalaxyV2.emergencyWithdraw confirmed (to zero)   Block: 32   Gas used: 23263 (0.19%)

Transaction sent: 0xcd98c787f600863b624b5b9e29f118ea65e8378b7aa05e97de819e5eebf0e9b0
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 17
  BlinkGalaxyV2.emergencyWithdraw confirmed (no tokens)   Block: 33   Gas used: 25754 (0.21%)

Transaction sent: 0x4fc99fc3c3ea6dc273676ad495ba372c6025fe35be820d7d7a5a102c0aaecb03
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 8
  ERC20Mock.mint confirmed   Block: 34   Gas used: 65821 (0.55%)

Transaction sent: 0x66d3491f79d782971b708e0010037c949e85cbd77eb6ba93be611bb1d01d3741
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 18
  BlinkGalaxyV2.emergencyWithdraw confirmed   Block: 35   Gas used: 41836 (0.35%)

tests/test_blink_galaxy.py::test_emergency_withdraw PASSED
tests/test_blink_galaxy.py::test_delegate RUNNING
Transaction sent: 0x58394a4b4323a45c033955d4e03dfedb0ce7cf8f4819a71f7cce4c8f0405e566
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 9
  BlinkGalaxyV2.constructor confirmed   Block: 36   Gas used: 2050922 (17.09%)
  BlinkGalaxyV2 deployed at: 0xa3B53dDCd2E3fC28e8E130288F2aBD8d5EE37472

Delegator addr: 0x0063046686E46Dc6F15918b61AE2B121458534a5
to Delegate addr: 0x21b42413bA931038f35e7A5224FaDb065d297Ba3
Transaction sent: 0xa549680e298687efed3c4c54885712f4fc95f16bd10f4f103a44bd9e2d70853f
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 5
  BlinkGalaxyV2.delegate confirmed   Block: 37   Gas used: 45699 (0.38%)

delegate TX: {'DelegateChanged': [OrderedDict([('delegator', '0x0063046686E46Dc6F15918b61AE2B121458534a5'), ('fromDelegate', '0x0000000000000000000000000000000000000000'), ('toDelegate', '0x21b42413bA931038f35e7A5224FaDb065d297Ba3')])]}

Transaction sent: 0x9d51f4cbebb488d3fa883ac230e2110403236c83394c8d157b64a245588552fe
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 19
  BlinkGalaxyV2.mint confirmed   Block: 38   Gas used: 70631 (0.59%)

mint TX: {'Transfer': [OrderedDict([('from', '0x0000000000000000000000000000000000000000'), ('to', '0x0063046686E46Dc6F15918b61AE2B121458534a5'), ('value', 500000000000000000)])], 'Minted': [OrderedDict([('minter', '0x33A4622B82D4c04a53e170c638B944ce27cffce3'
), ('receiver', '0x0063046686E46Dc6F15918b61AE2B121458534a5'), ('mintAmount', 500000000000000000)])]}

Get Current votes delegator (0x0063046686E46Dc6F15918b61AE2B121458534a5): 0
Get Current votes toDelegate (0x21b42413bA931038f35e7A5224FaDb065d297Ba3): 0
tests/test_blink_galaxy.py::test_delegate FAILED
```

Annexes

Testing code:

```
from brownie import (
    reverts,
)


from scripts.helpful_scripts import (
    ZERO_ADDRESS,
    DAY_TIMESTAMP,
    get_account,
    get_timestamp,
    get_chain_number,
    increase_timestamp
)


from scripts.deploy import (
    deploy_erc,
    deploy_blink_galaxy,
)


def test_constructor(only_local):
    # Arrange
    owner = get_account(0)
    multisig = get_account(1)


    with reverts("multisig zero"):
        deploy_blink_galaxy(owner, ZERO_ADDRESS)
```

```
    blink_token = deploy_blink_galaxy(owner, multisig)
    assert blink_token.cap() == 10_000_000_000 * 1e18


def test_set_whitelist(only_local):
    # Arrange
    owner = get_account(0)
    multisig = get_account(1)
    other = get_account(2)


    blink_token = deploy_blink_galaxy(owner, multisig)


    tx = blink_token.setWhitelistEnabled(False, {"from": multisig})
    assert tx.events['WhitelistToggled'][0]['enabled'] == False


    tx = blink_token.setWhitelisted(other, False, {"from": multisig})
    assert tx.events['WhitelistSet'][0]['account'] == other
    assert tx.events['WhitelistSet'][0]['allowed'] == False


    tx = blink_token.setWhitelisted(other, True, {"from": multisig})
    assert tx.events['WhitelistSet'][0]['account'] == other
    assert tx.events['WhitelistSet'][0]['allowed'] == True


    tx = blink_token.batchSetWhitelisted([other], False, {"from": multisig})
    assert tx.events['WhitelistSet'][0]['account'] == other
    assert tx.events['WhitelistSet'][0]['allowed'] == False


def test_mint(only_local):
```

```
    # Arrange
    owner = get_account(0)
    multisig = get_account(1)
    other = get_account(2)
    another = get_account(3)

    blink_token = deploy_blink_galaxy(owner, multisig)

    with reverts("Cap exceeded"):
        blink_token.mint(other, 10_000_000_001 * 1e18, {"from": multisig})

    tx = blink_token.mint(other, 10 * 1e18, {"from": multisig})
    assert tx.events['Minted'][0]['minter'] == multisig
    assert tx.events['Minted'][0]['receiver'] == other
    assert tx.events['Minted'][0]['mintAmount'] == 10 * 1e18

    tx = blink_token.mint(another, 5 * 1e18, {"from": multisig})
    assert tx.events['Minted'][0]['minter'] == multisig
    assert tx.events['Minted'][0]['receiver'] == another
    assert tx.events['Minted'][0]['mintAmount'] == 5 * 1e18

    blink_token.setWhitelistEnabled(True, {"from": multisig})

    with reverts("to not whitelisted"):
        blink_token.mint(other, 1 * 1e18, {"from": multisig})


def test_burn(only_local):
```

```
    # Arrange
    owner = get_account(0)
    multisig = get_account(1)
    other = get_account(2)
    another = get_account(3)

    blink_token = deploy_blink_galaxy(owner, multisig)

    # mint some tokens
    blink_token.mint(other, 10 * 1e18, {"from": multisig})
    blink_token.mint(another, 5 * 1e18, {"from": multisig})

    tx = blink_token.burn(1e18, {"from": other})
    assert tx.events['Burned'][0]['burner'] == other
    assert tx.events['Burned'][0]['burnAmount'] == 1 * 1e18

    tx = blink_token.burn(1e18, {"from": another})
    assert tx.events['Burned'][0]['burner'] == another
    assert tx.events['Burned'][0]['burnAmount'] == 1 * 1e18

    blink_token.setWhitelistEnabled(True, {"from": multisig})

    with reverts("from not whitelisted"):
        blink_token.burn(1e18, {"from": another})

def test_transfer(only_local):
    # Arrange
    owner = get_account(0)
```

```
    multisig = get_account(1)
    other = get_account(2)
    another = get_account(3)


    blink_token = deploy_blink_galaxy(owner, multisig)
    # mint some tokens
    blink_token.mint(other, 10 * 1e18, {"from": multisig})
    blink_token.mint(another, 5 * 1e18, {"from": multisig})


    tx = blink_token.transfer(another, 1e18, {"from": other})
    assert tx.events['Transfer'][0]['from'] == other
    assert tx.events['Transfer'][0]['to'] == another
    assert tx.events['Transfer'][0]['value'] == 1e18


    tx = blink_token.transfer(owner, 1e18, {"from": other})
    assert tx.events['Transfer'][0]['from'] == other
    assert tx.events['Transfer'][0]['to'] == owner
    assert tx.events['Transfer'][0]['value'] == 1e18


    blink_token.setWhitelistEnabled(True, {"from": multisig})


    with reverts("transfer not whitelisted"):
        blink_token.transfer(another, 1e18, {"from": other})


    blink_token.batchSetWhitelisted([other, another], True, {"from": multisig})


    tx = blink_token.transfer(another, 1e18, {"from": other})
    assert tx.events['Transfer'][0]['from'] == other
```

```
    assert tx.events['Transfer'][0]['to'] == another
    assert tx.events['Transfer'][0]['value'] == 1e18


def test_emergency_withdraw(only_local):
    # Arrange
    owner = get_account(0)
    multisig = get_account(1)
    other = get_account(2)
    another = get_account(3)


    mock_token = deploy_erc(owner, "test", "test")
    blink_token = deploy_blink_galaxy(owner, multisig)


    with reverts("to zero"):
        blink_token.emergencyWithdraw(mock_token.address, ZERO_ADDRESS, {"from": multisig})
    with reverts("no tokens"):
        blink_token.emergencyWithdraw(mock_token.address, other, {"from": multisig})


    # mit some tokens
    mock_token.mint(blink_token.address, 10e18)
    tx = blink_token.emergencyWithdraw(mock_token.address, other, {"from": multisig})
    assert tx.events['Transfer'][0]['from'] == blink_token.address
    assert tx.events['Transfer'][0]['to'] == other
    assert tx.events['Transfer'][0]['value'] == 10e18


def test_delegate(only_local):
    # Arrange
    owner = get_account(0)
```

```
    multisig = get_account(1)

    other = get_account(2)

    another = get_account(3)

    extra = get_account(4)


    blink_token = deploy_blink_galaxy(owner, multisig)


    print(f"Delegator addr: {other}")

    print(f"to Delegate addr: {another}")


    tx = blink_token.delegate(another, {"from": other})

    print(f"delegate TX: {tx.events}\n")

    assert tx.events['DelegateChanged'][0]['delegator'] == other

    assert tx.events['DelegateChanged'][0]['fromDelegate'] == ZERO_ADDRESS

    assert tx.events['DelegateChanged'][0]['toDelegate'] == another


    tx = blink_token.mint(other, 5e18, {"from": multisig})

    print(f"mint TX: {tx.events}\n")

    assert tx.events['Minted'][0]['minter'] == multisig

    assert tx.events['Minted'][0]['receiver'] == other

    assert tx.events['Minted'][0]['mintAmount'] == 5e18


    print(f"Get Current votes delegator ({other}): {blink_token.getCurrentVotes(other)}")

    print(f"Get Current votes toDelegate ({another}):
{blink_token.getCurrentVotes(another)}")


    assert blink_token.getCurrentVotes(another) != 0
```

Technical Findings Summary

Findings

Vulnerability Level	Total	Pending	Not Apply	Acknowledged	Partially Fixed	Fixed
High	1			1		
Medium	1			1		
Low	0					
Informational	0					

Assessment Results

Score Results

Review	Score
Global Score	**85/100**
Assure KYC	Not completed
Audit Score	85/100

The Following Score System Has been Added to this page to help understand the value of the audit, the maximum score is 100, however to attain that value the project must pass and provide all the data needed for the assessment. Our Passing Score has been changed to 84 Points for a higher standard, if a project does not attain 85% is an automatic failure. Read our notes and final assessment below. The Global Score is a combination of the evaluations obtained between having or not having KYC and the type of contract audited together with its manual audit.

Audit PASS

Following our comprehensive security audit of the token contract for the Blink Galaxy project, we inform you that the project has met the necessary security standards.

Disclaimer

Assure Defi has conducted an independent security assessment to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the reviewed code for the scope of this assessment. This report does not constitute agreement, acceptance, or advocating for the Project, and users relying on this report should not consider this as having any merit for financial Blink Galaxy in any shape, form, or nature. The contracts audited do not account for any economic developments that the Project in question may pursue, and the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude, and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are entirely free of exploits, bugs, vulnerabilities or deprecation of technologies.

All information provided in this report does not constitute financial or investment Blink Galaxy, nor should it be used to signal that any person reading this report should invest their funds without sufficient individual due diligence, regardless of the findings presented. Information is provided ‘as is, and Assure Defi is under no covenant to audit completeness, accuracy, or solidity of the contracts. In no event will Assure Defi or its partners, employees, agents, or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions or actions with regards to the information provided in this audit report.

The assessment Blink Galaxys provided by Assure Defi are subject to dependencies and are under continuing development. You agree that your access or use, including but not limited to any Blink Galaxy reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies with high levels of technical risk and uncertainty. The assessment reports could include false positives, negatives, and unpredictable results. The Blink Galaxy may access, and depend upon, multiple layers of third parties.