Assure DeFi®

THE VERIFICATION GOLD STANDARD

# Security Assessment

# MQSALA

Date: 29/06/2025

Audit Status: PASS

Audit Edition: Advanced+

ASSURE DEFI®

THE VERIFICATION GOLD STANDARD

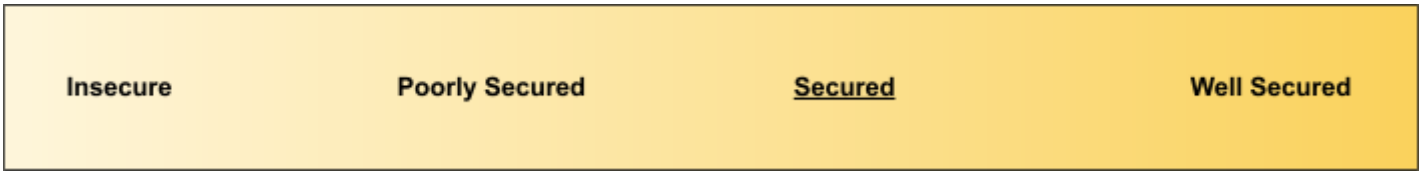
# Risk Analysis

## 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| Low | Low-level vulnerabilities are primarily associated with outdated or unused code snippets that generally do not significantly impact execution, sometimes they can be ignored.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 MQSALA 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://repo.sourcify.dev/97/0x58805656ECC5ACBA962Fbb955bDb57A88214aA7A<br><br>Final version:<br><br>https://repo.sourcify.dev/97/0x8DA6bD7b65bb6b3D5521205ac3E3B781a2ab44cB |
| **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.<br>• Race conditions and Reentrancy. Cross-function race conditions.<br>• Possible delays in data delivery.<br>• Oracle calls.<br>• Front running.<br>• Timestamp dependence.<br>• Integer Overflow and Underflow.<br>• DoS with Revert.<br>• DoS with block gas limit.<br>• Methods execution permissions.<br>• Economy model.<br>• Private user data leaks.<br>• Malicious Event log.<br>• Scoping and Declarations.<br>• Uninitialized storage pointers.<br>• Arithmetic accuracy.<br>• Design Logic.<br>• Cross-function race conditions.<br>• Safe Zeppelin module.<br>• Fallback function security.<br>• Overpowered functions / Owner privileges |

# AUDIT OVERVIEW


**1. Unprotected Initializer [FIXED ✅]**

**Issue**: The initialize() function can be called by anyone until it is initialized, allowing a front-running attacker to initialize the contract with malicious parameters.

**Recommendation**: Use OpenZeppelin initializer modifier on initialize() and add a disableInitializers() call in the implementation contract constructor

**Fix**: Added a constructor() { _disableInitializers(); } and marked initialize(.) with initializer, so nobody can re-init after deployment.

**2. Voting Power Manipulation [FIXED ✅]**

**Issue**: Inherits from ERC20Votes without restricting delegation, a malicious actor could delegate votes before token transfers, skewing governance power.

**Recommendation**: Restrict delegation functionality until licenseObtained == true (for example override delegation functions to enforce this).

**Fix**: The _delegate() function is overridden to require licenseObtained == true

**3. Asymmetric Address Restriction [FIXED ✅]**

**Issue**: Restricted addresses can receive tokens but cannot send them, leading to potential token traps, compliance bypasses, or fund lockups.

**Recommendation**: Enforce full blacklisting by blocking both inbound and outbound transfers:

```
if (from != address(0) && to != address(0)) {  // Skip mint/burn
    require(!restricted[from], "MQSLA: sender restricted");
    require(!restricted[to],   "MQSLA: recipient restricted");  // Add this
}
```

**Fix**: The _update() function checks both sender and receiver restrictions

## MEDIUM

### 1. Inefficient Burn Mechanism [FIXED ✅]

**Issue**: The mint function burns tokens by minting to address(this) and then burning them, which is inefficient and may fail if the contract is restricted.

**Recommendation**: Directly call _burn(address(0), burnAmt) (or integrate burn logic into _mint) to eliminate the intermediate step.

**Fix**: Now directly mints to address(0) instead of intermediate steps

### 2. Potential Reentrancy in _update [FIXED ✅]

**Issue**: The custom transfer-restriction logic in _update runs before calling super._update(), which could open a reentrancy vector if any overridden hooks perform external calls

**Recommendation**: Reorder to follow checks-effects-interactions: do all checks and state updates first, then invoke super._update() last

**Fix**: The function follows Checks-Effects-Interactions pattern

### 3. Auto-Burn Calculation Error [FIXED ✅]

**Issue**: Uses integer division for burnAmt = amount * burnBP / 10_000, so small mints (for example amount = 1, burnBP = 1) result in burnAmt = 0, bypassing the burn logic.

**Recommendation**: Switch to a rounding-up formula, e.g. burnAmt = (amount * burnBP + 10_000 - 1) / 10_000, to ensure even small amounts incur the intended burn.

**Fix**: The contract now uses a ceiling division formula to ensure even small amounts are burned correctly

### 4. Whitelist Bypass Potential [FIXED ✅]

**Issue**: Transfer restrictions skip when from == address(0) or to == address(0) (mint/burn), potentially allowing restricted addresses to receive tokens.

**Recommendation**: Decide whether mint/burn paths should enforce whitelist/restriction checks and update _update accordingly.

**Fix**: Added explicit checks for minting to restricted addresses

## LOW

### 1. TGE Timestamp Unused [FIXED ✅]

**Issue**: The tgeTimestamp variable is set during initialization but never referenced in any contract logic.

**Recommendation**: Either remove tgeTimestamp if it's not needed or add the intended time-based restrictions (for example, lock transfers until after tgeTimestamp).

**Fix**: The unused tgeTimestamp variable has been removed entirely

## INFORMATIONAL

### 1. Upgrade-Admin Centralization [Acknowledge]

**Issue**: Because this is UUPS, onlyRole(ADMIN_ROLE) can upgrade your implementation. If that key is compromised or mis-managed, an attacker can swap in malicious logic.

**Recommendation**: Consider a time-lock or multisig on the upgrade step in your on-chain governance.

If you are using a Gnosis Safe or similar, ensure the ADMIN_ROLE account is a safe multisig.

# 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_mqsla_token.py::test_transfer RUNNING
Transaction sent: 0xb3345cba699acd57539a5023ab6ac26854ef46e472c206e5be33e54ce3be4df7
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  MQSLAToken.constructor confirmed   Block: 1   Gas used: 3245799 (27.05%)
  MQSLAToken deployed at: 0x3194cBDC3dbcd3E11a07892e7bA5c3394048Cc87

Transaction sent: 0xea1c3f0a53ff653a0c4395f1cbbbecc46dfa1a45a3c2f2e05a346b953c477fd2
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  MQSLAToken.initialize confirmed   Block: 2   Gas used: 285787 (2.38%)

Transaction sent: 0x14fd45afbb0b0e2e966959d7d0a30cf8b58c45b26a8e8b1ced251b06c1154bc7
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 2
  MQSLAToken.mint confirmed   Block: 3   Gas used: 116494 (0.97%)

Transaction sent: 0x3c88e450da622272ef08921befeda931a55bada80bf0b5844020a53d3bc2ce87
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 3
  MQSLAToken.mint confirmed (MQSLA: cap)   Block: 4   Gas used: 24231 (0.20%)

Transaction sent: 0xc47376836ce7a2cbd4c8f881c94ccdeeab5994f5cdbfb52fd1af0dbf4bbad041
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 4
  MQSLAToken.setBurnBP confirmed (MQSLA: burn too high)   Block: 5   Gas used: 22831 (0.19%)

Transaction sent: 0xc410130eac57f3eb8b8d67bbe7361d640a5464e09166b3867fa385d4af7ef7d4
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 5
  MQSLAToken.setBurnBP confirmed   Block: 6   Gas used: 44630 (0.37%)

Transaction sent: 0x72f8bca2150d72ddc14ff2049e89f69b1028a4d372e6093fb03383c0030fc55b
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 6
  MQSLAToken.mint confirmed   Block: 7   Gas used: 119032 (0.99%)

Transaction sent: 0xe2b0947001eef7a672ea2f7ac4e8408ddb3552195fd7561fa183df74eb17bba2
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  MQSLAToken.transfer confirmed (MQSLA: transfers locked pre-licence)   Block: 8   Gas used: 25887 (0.22%)

Transaction sent: 0xfec394075fd9b705cee17d2d4ff674fb2d2471fd170ac8fb7bc8c13df791ff55
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 7
  MQSLAToken.setWhitelist confirmed   Block: 9   Gas used: 44115 (0.37%)

Transaction sent: 0x308986bd293e0cf62d99acdc580b705a1f2eef132827c32d318c2aed710f82dc
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  MQSLAToken.transfer confirmed   Block: 10   Gas used: 57095 (0.48%)

Transaction sent: 0x8ed096d1da093163fd1827f394d90e98be269f24eae56ddeaf88dc9925f5be10
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 8
  MQSLAToken.setRestricted confirmed   Block: 11   Gas used: 45585 (0.38%)

Transaction sent: 0x2461b573f1c36427de70ff1cde3dfac5b2d24a5647f97b309a0e292bfe8f60a8
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 9
  MQSLAToken.setLicenseObtained confirmed   Block: 12   Gas used: 29684 (0.25%)

Transaction sent: 0x2e6a85c53bf71b5048c000abce87d59206b94a02289f0f65f949490232725ee0
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  MQSLAToken.transfer confirmed (MQSLA: sender restricted)   Block: 13   Gas used: 24964 (0.21%)

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

```
tests/test_mqsla_token.py::test_burn_from RUNNING
Transaction sent: 0x2d44e36e67abe14045a4d5baadb6933c9039660c860b1802131e66a4dcb1990d
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 10
  MQSLAToken.constructor confirmed   Block: 14   Gas used: 3245799 (27.05%)
  MQSLAToken deployed at: 0xb6286fAFd0451320ad6A8143089b216C2152c025

Transaction sent: 0xa1893fd463134207d0676a39b4570c11a43e02c875b7852008ddf2c92b0129ec
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 11
  MQSLAToken.initialize confirmed   Block: 15   Gas used: 285787 (2.38%)

Transaction sent: 0x81900e00075e0185e9395cab75d73a27c243d870bfbcc555e356a5168a195719
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 12
  MQSLAToken.mint confirmed   Block: 16   Gas used: 116494 (0.97%)

Transaction sent: 0xe702f124891cdb25475419aa658b77c2658c0d9c30f735f4d7744267c42a5222
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 13
  MQSLAToken.mint confirmed   Block: 17   Gas used: 88581 (0.74%)

Transaction sent: 0x76eaa7112d7ef551018305de992686eada09bf692c146e2e65029f7a52bac862
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 14
  MQSLAToken.mint confirmed   Block: 18   Gas used: 88569 (0.74%)

Transaction sent: 0xd5572d1bffac32d3056ca635ef04383fcdf84c33654b5aea36ab3fde87e9d83b
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 2
  MQSLAToken.approve confirmed   Block: 19   Gas used: 44368 (0.37%)

Transaction sent: 0xbce329c2c1f2fc94f121d48074319e830a79f515a0146bc9d509dbdb598cf6ea
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 15
  MQSLAToken.burnFrom confirmed   Block: 20   Gas used: 62227 (0.52%)

tests/test_mqsla_token.py::test_burn_from PASSED
```

# 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_token
)

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

    token = deploy_token(owner)
    token.initialize("Token", "T", get_timestamp(180), owner)

    tx = token.mint(other, 100e18, {"from": owner})
    assert tx.events['Transfer'][0]['from'] == ZERO_ADDRESS
    assert tx.events['Transfer'][0]['to'] == other
```

```
    assert tx.events['Transfer'][0]['value'] == 100e18

    with reverts("MQSLA: cap"):
        token.mint(other, 1000000001e18, {"from": owner})
    with reverts("MQSLA: burn too high"):
        token.setBurnBP(3000, {"from": owner})

    tx = token.setBurnBP(1000, {"from": owner})
    assert tx.events['BurnRateChanged'][0]['newBP'] == 1000

    tx = token.mint(owner, 1000e18, {"from": owner})
    assert tx.events['Transfer'][2]['from'] == ZERO_ADDRESS
    assert tx.events['Transfer'][2]['to'] == owner
    assert tx.events['Transfer'][2]['value'] == 900e18

    # test transfers
    with reverts("MQSLA: transfers locked pre-licence"):
        token.transfer(extra, 1e18, {"from": other})

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

    tx = token.setRestricted(extra, True, {"from": owner})
    assert tx.events['AddressRestriction'][0]['acct'] == extra
    assert tx.events['AddressRestriction'][0]['restricted'] == True

    token.setLicenseObtained(True, {"from": owner})

    with reverts("MQSLA: sender restricted"):
        token.transfer(another, 1e18, {"from": extra})
```

```python
def test_burn_from(only_local):
    # Arrange
    owner = get_account(0)
    other = get_account(1)
    extra = get_account(2)

    token = deploy_token(owner)
    token.initialize("Token", "T", get_timestamp(180), owner)
    # mint tokens
    token.mint(owner, 1000e18, {"from": owner})
    token.mint(other, 100e18, {"from": owner})
    token.mint(extra, 100e18, {"from": owner})

    token.approve(owner, 10e18, {"from": other})
    tx = token.burnFrom(other, 10e18, {"from": owner})
    assert tx.events['Transfer'][0]['from'] == other
    assert tx.events['Transfer'][0]['to'] == ZERO_ADDRESS
    assert tx.events['Transfer'][0]['value'] == 10e18
```

# Technical Findings Summary

## Findings

| Vulnerability Level | Total | Pending | Not Apply | Acknowledged | Partially Fixed | Fixed |
|---|---|---|---|---|---|---|
| High | 3 | | | | | 3 |
| Medium | 4 | | | | | 4 |
| Low | 1 | | | | | 1 |
| Informational | 1 | | | 1 | | |

# Assessment Results

## Score Results

| Review | Score |
|---|---|
| **Global Score** | **90/100** |
| Assure KYC | Not completed |
| Audit Score | 90/100 |

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 MQSALA 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 adMQSALA 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 adMQSALA, 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 serMQSALAs 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 serMQSALAs, 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 serMQSALAs may access, and depend upon, multiple layers of third parties.